N° d'ordre 117
Année 2009

Université de Franche-Comté
Sciences, Techniques et Gestion de l'Industrie

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE FRANCHE-COMTÉ

Spécialité : **Informatique**

Interactivité et Disponibilité des Données dans les Systèmes Multimédias Distribués

par **Husam ALUSTWANI**

Soutenue le 25 Juin 2009 devant la commission d'examen :

Rapporteurs	S. BENBERNOU	Maître de conférences, Université de Lyon I
	K. YETONGNON	Professeur, Université de Dijon
Directeur de Thèse	J. M. BAHI	Professeur, Université de Franche-Comté
Co-directeur de Thèse	A. MOSTEFAOUI	Maître de conférences, Université de Franche-Comté
Examinateurs	L. BRUNIE	Professeur, INSA de Lyon
	H. GUYENNET	Professeur, Université de Franche-Comté

À mes parents.

Remerciements

C'est avec un grand plaisir que je réserve ces lignes en témoignage de ma reconnaissance à tous ceux qui m'ont accompagné, soutenu et aidé pour accomplir les travaux présentés dans ce manuscrit.

Tout d'abord Jacques M. Bahi, mon directeur de thèse, qui a bien accepté de superviser mes travaux. Ensuite, Ahmed Mostefaoui, qui a encadré mes travaux dans cette thèse. Je tiens à lui exprimer ma gratitude pour les pistes de réflexion qu'il m'a fait prendre.

J'adresse aussi mes remerciements à Mme. Salima Benbernou et M. Kokou Yetongnon pour m'avoir fait l'honneur d'accepter d'être rapporteurs de cette thèse. Que M. Lionel Brunie et M. Hervé Guyennet soient également remerciés pour avoir accepté d'être membres de mon jury.

Je suis très reconnaissant envers Pierre Comte, Sébastien Miquée et Michel Salomon qui m'ont chaleureusement proposé de relire ce document. Je les remercie pour toutes les rectifications qu'ils ont apportées, et qui m'ont permis d'améliorer la qualité linguistique de ce manuscrit.

Un grand merci à l'ensemble des membres de l'équipe Algorithmique Numérique Distribuée du Laboratoire d'Informatique de l'Université de Franche-Comté pour les moments de détente et de discussion tellement agréables. Je remercie tout particulièrement Michel, Jean-Luc, Laurence, Raphaël et David pour leur aide mais surtout pour l'ambiance de travail toujours joviale que vous avez su créée au sein de l'équipe.

Je ne manquerais pas de remercier mes collègues de bureau, Mirna et Sébastien pour avoir supporté mes bavardages durant ces dernières années.

Enfin, je tiens aussi à remercier mes parents pour leur soutien inconditionnel, et surtout mon père qui n'a jamais cessé de m'encourager de si loin, notamment lorsque le moral tomba au plus bas.

À tous, mon infinie gratitude ...

TABLE DES MATIÈRES

Remerciements **v**

Table des Matières **vii**

Liste des Tableaux **xi**

Liste des Figures **xiii**

Introduction **1**

1 Systèmes de streaming distribués : vue d'ensemble **7**

1.1 Les données audiovisuelles 9

1.1.1 Caractéristiques 9

1.1.2 Encodage 11

1.1.3 Formats audiovisuels 15

1.2 Streaming audiovisuel 16

1.2.1 Streaming versus téléchargement 16

1.2.2 Avantages 18

1.2.3 Modes 19

1.2.4 Applications 20

1.3 Système de streaming 22

1.3.1 Lecteur audiovisuel 22

1.3.2 Réseaux et streaming 25

1.3.3 Serveur de streaming 28

1.3.4 « Client pull » versus « serveur push » 42

1.4 Les caches dans les systèmes de streaming 44
1.4.1 Cache de lecture 44
1.4.2 Cache de serveur 45
1.4.3 Caches de réseau (« Proxy ») 47
1.5 Streaming distribué 52
1.5.1 Serveurs sur grappe de machines 53
1.5.2 Serveur sur grille 55
1.5.3 Streaming collaboratif 56

2 Interactivité dans les présentations multimédias 59
2.1 Présentations multimédias 61
2.1.1 Composition 61
2.1.2 Synchronisation durant une présentation 63
2.1.3 Spécifications et environnements d'édition 64
2.1.4 Systèmes de streaming multimédia 67
2.1.5 Navigation dans les présentations multimédias 70
2.2 Parcours rapide dans les présentations multimédias 75
2.2.1 Problématique 75
2.2.2 Parcours rapide proposé 76
2.2.3 Mise en œuvre 79
2.2.4 Politique de préchargement : CPS 80
2.3 Étude de performance 86
2.3.1 Scénario de test 87
2.3.2 Résultats 88
2.3.3 Discussion 93
2.4 Conclusion 94

3 Disponibilité des données dans les systèmes de streaming P2P 95
3.1 Systèmes pair-à-pair 97
3.1.1 Définitions 97
3.1.2 Caractéristiques 98
3.1.3 Taxinomie des architectures 100
3.1.4 Domaines d'applications 104
3.2 Systèmes de streaming P2P 106
3.2.1 Architectures hybrides 107
3.2.2 Architecture pair-à-pair 110

3.3 Disponibilité de données dans les systèmes de streaming P2P 111
3.3.1 Description du problème 112
3.3.2 Mise en cache des suffixes 116
3.4 Étude de performances 125
3.4.1 Scénarios de tests 125
3.4.2 Résultats 127
3.4.3 Discussion 130
3.5 Conclusion 131

4 JStreaper : un système de streaming Pair-à-Pair 133
4.1 Architecture logicielle 135
4.1.1 Scénario d'utilisation 136
4.1.2 Modèle de communication 136
4.1.3 Dispatcher 136
4.1.4 Streaper 137
4.2 Choix d'implémentation 139
4.2.1 Technologie Java 139
4.2.2 Java RMI 141
4.2.3 Java Media Framework (JMF) 141
4.3 Réalisation 143
4.3.1 Dispatcher 143
4.3.2 Player 145
4.3.3 Streamer 146
4.4 Tests préliminaires 148
4.5 Conclusion 149

Conclusion 151

Bibliographie personnelle 157

Glossaire 159

Bibliographie 176

Résumé 176

LISTE DES TABLEAUX

1.1 Volume des différents types de données. 10
1.2 Principaux standards de codage audiovisuel. 10

2.1 Récapitulatif des valeurs par défaut des différents paramètres considérés. 88

3.1 Plusieurs tailles de suffixes et leurs instants de début en fonction de ***Pcache***, calculés en utilisant l'équation (3.1). 119
3.2 Valeurs par défaut des différents paramètres considérés dans les expérimentations. 126

LISTE DES FIGURES

1.1 Isochronisme dans une séquence vidéo. 9
1.2 Techniques de découpage de données audiovisuelles VBR. 11
1.3 Son numérisé à différentes fréquences d'échantillonnage représenté pour différents nombres de bits. 12
1.4 Ordre de codage dans le modèle IPB de la norme MPEG-1. 14
1.5 Hiérarchisation spatiale et temporelle. 15
1.6 Téléchargement versus streaming 17
1.7 Diffusion individuelle de vidéos à la demande. 19
1.8 Multidiffusion d'un événement en direct. 20
1.9 Système de streaming. 22
1.10 Composants d'un lecteur audiovisuel. 24
1.11 Classement des protocoles de streaming dans les couches du modèle OSI. 26
1.12 Composants d'un serveur de streaming. 29
1.13 Fonctionnalités d'un serveur de streaming. 31
1.14 Géométrie d'un disque dur magnétique. 36
1.15 Algorithmes d'ordonnancement d'accès aux données placées sur un disque. 38
1.16 « Client pull » versus « serveur push ». 43
1.17 Les caches dans un système de streaming. 44
1.18 Distance entre des streams du point de vue de la mémoire. 47
1.19 Architecture à trois niveaux : serveur-proxy-client. 48
1.20 Politiques de suppression optimisées pour des contenus encodés en multi-couche. 50
1.21 Découpage d'un contenu audiovisuel en segments. 51
1.22 Proxies coopérant pour acheminer les différentes descriptions d'un contenu audiovisuel aux clients via des chemins différents. 51

1.23 Serveur en grappe, architecture à deux tiers. 54
1.24 Serveur en grappe, architecture flat. 55
1.25 Serveur en grille. 56

2.1 Structure logique d'une présentation. 62
2.2 Disposition spatiale d'une présentation. 62
2.3 Scénario temporel d'une présentation. 63
2.4 Synchronisation à grain fin entre une vidéo et une piste audio. 64
2.5 Lecteur de présentations multimédias. 69
2.6 Les trois types d'interaction dépendante du document. 71
2.7 Lecture accélérée, à coefficient 2, d'une vidéo cadencée à 15 images/seconde. . . . 72
2.8 Accélération simulée d'une vidéo cadencée à 15 images/seconde en sautant une image sur deux. 72
2.9 L/MRP : exemple d'ensembles d'interaction avec les valeurs de pertinence associées aux blocs. 73
2.10 Synopsis d'une vidéo. 74
2.11 Exemple d'idées dans une présentation multimédia. 77
2.12 Parcours rapide d'une présentation multimédia au travers des points d'interaction. 78
2.13 Décomposition d'une idée en unités de présentation. 80
2.14 Paraboles de base, utilisées dans les fonctions de pertinence. 82
2.15 Fonctions de pertinence dans la politique CPS. 83
2.16 CPS : exemple d'unités à préchargées avec une valeur donnée de β et plusieurs valeurs de α. 84
2.17 Scénario d'une présentation de test avec 30 vidéos qui se succèdent les unes aux autres. 87
2.18 CPS : impact de la taille du cache. 89
2.19 CPS : impact du seuil de préchargement β. 90
2.20 CPS : impact du paramètre α. 91
2.21 CPS : impact de la fréquence des interactions. 92

3.1 Vue simplifiée du modèle client-serveur versus le modèle pair-à-pair. 98
3.2 Exemple d'architecture pseudo pair-à-pair. 100
3.3 Exemple d'architecture pseudo pair-à-pair, où l'annuaire est réparti entre plusieurs super-nœuds. 101
3.4 Exemple d'un graphe formé par l'interconnexion des pairs. 102
3.5 Topologie arborescente d'un système de streaming P2P hybride avec dix pairs hiérarchisés en trois niveaux. 107

3.6 Reconstruction d'un arbre de diffusion suite au départ du Pair 0. 108
3.7 Une partie de l'arborescence d'un système de streaming P2P hybride dans une application MOD, avec les instants de connexion des pairs. 108
3.8 Un système de streaming P2P hybride avec deux arbres de diffusion, chacun correspondant à une description du même contenu. 109
3.9 Topologie maillée dans un système de streaming P2P. 110
3.10 Collaboration entre deux pairs dans la transmission de blocs de données à un troisième. 111
3.11 Vue générale d'un système de streaming P2P. Les pairs A, B, C et D se connectent au système. Le pair D recherche une vidéo à regarder à travers le système (requête). Le pair C prend en charge le streaming de la vidéo demandée. 113
3.12 Suffixe d'une vidéo. 116
3.13 **Mise en cache des suffixes** : 1) Le Pair C (hôte) transmet une vidéo à un Pair D (client) en streaming ; 2) Le Pair D atteint le *point de déclenchement* T ; 3) Le Pair C lance une requête de suffixe au cache ; 4) La requête de suffixe est *acceptée* par le cache ; 5) Le Pair C commence à sauvegarder le suffixe de la vidéo dans le cache ; 6) La copie du suffixe est terminée et la requête de suffixe est maintenant *satisfaite* ; 7) Le cache est en mesure de transmettre le suffixe au Pair D, si le Pair C se déconnecte. 117
3.14 Fonctions de probabilité. 119
3.15 **Cache virtuel distribué (DVC)** : formé par l'agrégation des espaces de cache fournis par les pairs. 121
3.16 Le système asservi, avec tri des requêtes de suffixe. 123
3.17 Le système asservi, sans tri des requêtes de suffixes. 124
3.18 Impact du paramètre *Pcache* sur la disponibilité des données. 127
3.19 Impact de la taille du cache fourni par les pairs sur la disponibilité des données. . 128
3.20 Performances de notre stratégie adaptative en fonction de la taille du cache fourni par les pairs. 129
3.21 Comparaison entre FCFS, MDF et notre stratégie de gestion de cache adaptative. 130

4.1 Vue générale de l'architecture de JStreaper. 135
4.2 Architecture multithreadée dans un Streamer. 138
4.3 Fonctionnement de RMI. 141
4.4 Restitution de données audiovisuelles dans JMF. 142
4.5 Architecture d'un processeur JMF. 142
4.6 Application utilisant l'environnement JMF. 143
4.7 Synchronisation entre une vidéo et son discours audio dans un thread de réception. 145
4.8 Fenêtre de dialogue pour l'ouverture d'un stream. 145
4.9 Fenêtre principale d'un Player. 146

4.10 Fenêtre principale d'un Streamer. 147
4.11 Usage du CPU en fonction du nombre de threads simultanés dans un Streamer. . 148

Introduction

Depuis quelques années, nous assistons à une profusion de données audiovisuelles. La production de contenus multimédias est devenue accessible à tout public et à faible coût. De fait, nous sommes témoins de la démocratisation des dispositifs de restitution et de production de données audiovisuelles. À titre d'exemple, il est aujourd'hui courant de trouver dans un foyer plusieurs périphériques permettant de produire des vidéos : caméscopes, appareils photos numériques classiques ou intégrés à des téléphones mobiles, etc. Nous pouvons ainsi dire que nous vivons dans l'ère du multimédia.

Grâce à l'avancée spectaculaire de la puissance des ordinateurs grand public, la manipulation des données audiovisuelles est devenue très fréquente chez les particuliers. Par exemple, au lieu de simplement stocker les photos de vacances dans des répertoires datés, il est d'usage d'en faire des albums multimédias. Un album peut être constitué d'un simple défilement de photos liées sémantiquement ou chronologiquement à un sujet ou à un évènement particulier. Une musique de fond et/ou des annotations textuelles peuvent y être ajoutées pour créer un document composé selon le scénario souhaité. On parle alors de *présentation multimédia*. Plus généralement, une présentation multimédia est un regroupement de plusieurs médias, de différents types, qui sont synchronisés dans le temps.

Les présentations multimédias ont d'ores et déjà investies l'Internet, dans un spectre applicatif particulièrement large allant des boutiques virtuelles jusqu'à la télévision en ligne en passant par l'enseignement à distance. Quel que soit le domaine d'application, les exigences des utilisateurs, en terme de navigation dans de tels documents composés, augmentent constamment. En particulier, *le parcours rapide dans les présentations multimédias est aujourd'hui un enjeu très important, notamment lors d'un accès distant à celles-ci*. Cet enjeu constitue une des poblèmatiques auxquelles cette thèse apporte une réponse. À noter que les présentations multimédias posent des problèmes de gestion et de manipulation de données parmi les plus délicats. Ceci est dû d'une part à la nature isochrone et volumineuse des données audiovisuelles composant ces présentations, et d'autre part à la complexité de leur composition.

Par ailleurs, avec les progrès remarquables qu'ont enregistrés les supports de communication, il est de nos jours envisageable de partager facilement du contenu multimédia avec ses proches ou bien avec le monde entier. Une personne souhaitant partager un contenu, audiovisuel, fait

typiquement appel à des sites proposant un service d'hébergement de fichiers en ligne, comme RapidShare ou MediaFire par exemple. La personne transfère son fichier sur un serveur du prestataire choisi et récupère un lien identifiant ce fichier. Par le biais de ce lien, tout usager ayant le droit d'accès au fichier en question peut le télécharger puis le visionner. Le téléchargement peut être effectué en utilisant un protocole standard de transmission de fichiers tel que FTP.

Dans le mode d'accès aux contenus partagés décrit précédemment, la lecture d'un contenu n'est possible qu'après la fin de son téléchargement. Ce dernier implique un délai d'attente plus ou moins long en fonction du volume des données. Or, la consommation des contenus multimédias est marquée par le désir des utilisateurs d'y accéder instantanément, du moins avec un léger temps d'attente. En effet, un temps d'attente important est plus acceptable si le contenu est un logiciel, mais il l'est moins s'il s'agit d'un film ou d'une chanson. Toutefois, il existe un autre mode d'accès bien plus adapté à la transmission de contenus multimédias, à savoir la diffusion en continu, communément appelé *streaming*. Le streaming permet le téléchargement et la restitution simultanés de contenus multimédias. Ce mode permet donc d'éviter l'attente due à la récupération d'un contenu avant de pouvoir le visionner. La restitution du contenu peut alors commencer dès que les premières données sont rapatriées et elle se poursuit au fur et à mesure de l'arrivée des données. De plus, lorsque le mode de transmission est le streaming, les utilisateurs peuvent partager des contenus soumis à des droits d'auteurs (dont ils possèdent les licences), car ce mode d'accès permet de les visionner sans effectuer de copies locales.

Le streaming des données multimédias est actuellement en plein essor, car la demande de contenus audiovisuels en temps réel est en pleine croissance. Cet engouement a conjointement mobilisé les chercheurs et les industriels pour concevoir et mettre en place des systèmes permettant la diffusion en continu de données audiovisuelles. En une décennie, de nombreux prototypes et une multitude de serveurs commerciaux, comme YouTube ou DailyMotion, ont vu le jour. Ces serveurs se sont spécialisés dans le streaming audiovisuel, permettant ainsi aux utilisateurs de partager leurs contenus multimédias à une échelle mondiale. Les serveurs de streaming se sont alors trouvés face à l'un des défis qui ont accompagné l'Internet depuis sa naissance : la transmission simultanée de contenus à de nombreux utilisateurs, tout en évitant une augmentation des coûts proportionnels à la taille de l'auditoire.

Aujourd'hui, avec la démocratisation de terminaux dotés d'une grande puissance de traitement et disposant de connexions à haut débit, il est désormais possible que des « mini-serveurs » hébergés par ces terminaux s'attellent à la tâche de streaming. Suscités par le succès formidable des systèmes pair-à-pair (P2P), de nouveaux systèmes permettant le partage de contenus audiovisuels en mode streaming sont en train d'émerger. Fondés sur le concept de P2P, ces systèmes, dorénavant appelés *systèmes de streaming P2P*, rentrent progressivement dans les mœurs du grand public et sont promis à un bel avenir.

Dans un système de streaming P2P, les participants au réseau de diffusion, aussi dénommés *pairs*, peuvent prendre en charge les tâches de streaming, confiées auparavant à des serveurs dédiés. Toutefois, en l'absence de serveurs, la nature dynamique des pairs peut compromettre la disponibilité des contenus qu'ils partagent. *L'indisponibilité de données dans de tels systèmes reste à ce jour une problématique ouverte.* Elle constitue d'ailleurs une des barrières technologiques majeures au large déploiement de ces systèmes, et se trouve au cœur de cette thèse.

Contributions de la thèse

Les travaux présentés dans ce manuscrit portent sur l'amélioration des systèmes de streaming. Notre objectif est d'améliorer l'interactivité et la disponibilité des données au sein de ces systèmes. Nos contributions se déclinent en trois volets, résumés dans les paragraphes suivants.

Interactivité dans les présentations multimédias

Comme nous l'avons mentionné, une présentation multimédia est un regroupement de plusieurs médias, de différents types, qui sont synchronisés dans le temps. La puissance expressive offerte par ces présentations est à l'origine de leur grande popularité. Face à un tel engouement apparaît la nécessité de parcourir rapidement ces documents composés. Or, en raison de la multiplicité des objets dans une présentation, celle-ci peut devenir inintelligible si elle est accélérée de manière « conventionnelle » (accélération linéaire du temps). Avec une telle accélération, l'auditeur peut, en effet, ne pas percevoir certaines informations. Techniquement, la restitution accélérée d'un document composé ne soulève aucun défi si ce dernier est disponible localement. Dans le cas d'un accès distant en mode streaming, une telle restitution implique, en plus de l'incompréhension provoquée par l'accélération, une consommation excessive de ressources, en termes de bande passante réseau, de puissance de traitement et/ou de taille de mémoire vive. Le parcours rapide d'une présentation multimédia à distance soulève donc des problématiques tant au niveau conceptuel que technique, ce qui explique l'absence d'une telle fonctionnalité dans les systèmes de streaming actuels. En vertu de ces faits, la sémantique même de la navigation accélérée dans une présentation multimédia doit être révisée.

Dans ce contexte, nous avons défini une sémantique adéquate pour la navigation accélérée dans les présentations multimédias. L'originalité de notre approche réside dans le fait qu'elle relie les fonctionnalités d'avance rapide, et de recul rapide, d'une présentation à son contenu. Il s'agit de parcourir le document composé en passant de thème en thème. La nouvelle navigation accélérée se traduit donc par des sauts dans la présentation, d'une idée vers une autre, plutôt que d'accélérer la restitution. À l'issue d'un saut, l'usager doit attendre le rapatriement des données avant que celles-ci ne lui soient restituées. Dans le but de minimiser ce temps d'attente, nous avons opté pour l'utilisation d'un *cache* où les données sont chargées préalablement à leur restitution. Pour la gestion de ce cache, nous avons élaboré une heuristique de *préchargement* spécialement adaptée à la nouvelle sémantique du parcours rapide. Nous avons testé et validé nos propos au cours de nombreuses séries d'expérimentations par simulation. Les résultats obtenus montrent l'efficacité de notre heuristique quant à la réduction des temps de latence traditionnellement observés dans les systèmes de streaming, et par conséquent à l'amélioration de leur interactivité.

Disponibilité de données dans les systèmes de streaming P2P

Dans un système de streaming P2P, les usagers partagent leurs contenus multimédias en mode streaming. Comme cela a été mentionné, il est aujourd'hui possible que les pairs remplacent les serveurs de streaming en se chargeant de la diffusion de leurs contenus respectifs. La diffusion d'un contenu peut alors ne provenir que d'une seule source, le détenteur du dit contenu. Or, les utilisateurs de tels systèmes ont un comportement très volatil, dans le sens où ils peuvent rejoindre et quitter le système à tout moment. Le départ éventuel d'un usager, partageant un

contenu en cours d'accès par un autre, risque de mettre ce dernier dans l'impossibilité de terminer le visionnage du contenu en question. Ce problème d'indisponibilité de contenus en cours d'accès constitue l'une des limitations de ce type de système.

Dans ce cadre, nos contributions consistent en l'utilisation d'un cache, non pas pour dupliquer la totalité des contenus, mais uniquement les *suffixes* de ceux qui sont en cours d'accès. Nous nous restreignions également aux suffixes des contenus visionnés par des utilisateurs qui vont probablement continuer leur visionnage jusqu'au bout. Nos choix se justifient d'une part, par le manque de ressources nécessaires pour dupliquer la totalité des données, et d'autre part, par la volatilité des pairs. Notre objectif, précisons-le, n'est donc pas d'assurer une disponibilité totale des données, chose qui requiert beaucoup trop de ressources, mais plutôt de limiter les effets négatifs de la déconnexion fréquente des pairs, avec un coût raisonnable.

Dans un premier temps, nous avons utilisé un cache centralisé pour sauvegarder ces suffixes. Nous nous sommes appuyés sur un modèle probabiliste pour calculer la taille du suffixe à mettre en cache ainsi que l'instant où le processus de mise en cache doit être déclenché. Ensuite, nous avons étendu notre approche vers une architecture distribuée du cache, c'est-à-dire que chaque pair fournit un petit espace cache de telle sorte que l'agrégation de ces espaces forme un *cache virtuel distribué* (DVC). Les suffixes doivent alors être répartis de manière équitable afin d'éviter la surcharge de certains pairs. Nous avons donc mis au point une politique d'allocation qui distribue judicieusement les suffixes entre les pairs. De plus, nous avons élaboré une stratégie de gestion dynamique du DVC, qui adapte le volume global de données à mettre en cache en fonction de l'espace disponible dans le DVC. Cet espace est lui-même tributaire du nombre de pairs connectés et plus particulièrement de l'espace cache fourni par ces derniers.

Les résultats expérimentaux que nous avons obtenus montrent clairement que notre approche limite considérablement les effets de la déconnexion des pairs sur la disponibilité des données au sein d'un système de streaming P2P.

Un système de streaming P2P adaptatif : le prototype JStreaper

La plupart des systèmes de streaming P2P actuels sont conçus pour un déploiement à grande échelle, impliquant pour chaque contenu dans le système, la présence de plusieurs utilisateurs le partageant. Les usagers possédant un contenu donné coopèrent dans le streaming de celui-ci vers un utilisateur désirant le visionner. La collaboration consiste pour chacun à prendre en charge la transmission d'une partie des données désirées. Toutefois, la vulgarisation de l'informatique mobile a mis en relief de nouveaux « comportements » des utilisateurs. Typiquement, un usager se connecte pour regarder un ou plusieurs contenus, puis une fois le visionnage terminé, il se déconnecte. La durée de connexion des utilisateurs est donc souvent très courte voire imprévisible, on parle alors de comportement *volatil*. Ce type de comportement rend caduque l'hypothèse d'une connexion permanente et donc la disponibilité en de multiples exemplaires de tout contenu partagé.

Au travers de la conception et de la mise en œuvre de *JStreaper*, nous souhaitons tester la faisablité d'un système de streaming P2P où la transmission d'un contenu est assurée par un et un seul pair. À l'heure actuelle, les fournisseurs d'accès Internet n'offrent pas de connexion ADSL grand public permettant d'avoir la bande passante nécessaire pour assurer le streaming d'un contenu par un seul pair. Notre prototype trouve donc son domaine d'application dans les réseaux locaux à très haut débit. Cependant, cette limitation s'effacera avec la démocratisation

des connexions fibre optique, ce qui permettra l'utilisation de notre prototype à l'échelle d'Internet, sans aucune modification. Par ailleurs, les utilisateurs de JStreaper peuvent être équipés de terminaux plus ou moins puissants, en termes de capacités de traitement et/ou d'affichage, car notre prototype permet de redimensionner les images des vidéos transmises en fonction de la configuration du terminal récepteur (*adaptation spatiale*). Dans l'état actuel de nos connaissances, JStreaper est le premier prototype qui intègre le streaming P2P avec adaptation de contenus multimédias.

Plan du document

Cette thèse est structurée en 4 chapitres :

1. **Systèmes de streaming distribués - vue d'ensemble :** ce chapitre préliminaire est consacré à la présentation du domaine très vaste du multimédia, et en particulier les concepts de base du streaming audiovisuel. Nous y présentons notamment un état de l'art des techniques de gestion des serveurs vidéo. Nous décrivons l'évolution des architectures logicielles de ces serveurs en fonction des équipements les hébergeant, jusqu'au déploiement massif aujourd'hui basé sur le concept du pair-à-pair. Nous présentons également les divers types de cache dans un système de streaming, ainsi que leurs rôles respectifs.
2. **Streaming de présentations multimédias :** ce chapitre introduit les présentations multimédias et souligne les difficultés inhérentes à la gestion de tels documents composés. Nous passons en revue les techniques proposées dans la littérature pour naviguer au sein des présentations multimédias. Ensuite, nous exposons notre contribution qui relie les interactions de parcours rapide d'une présentation à son contenu et permet ainsi d'y naviguer en passant de thème en thème. Nous présentons également notre heuristique de préchargement, dont l'objectif est de minimiser le temps de latence, suite à une action de navigation accélérée. Nous concluons ce chapitre par une discussion des résultats obtenus suite aux expérimentations menées pour évaluer la pertinence de cette heuristique.
3. **Streaming Pair-à-Pair :** dans ce chapitre, nous faisons une introduction générale au concept du pair-à-pair, nous établissons également une taxinomie des architectures utilisées pour mettre en œuvre un système P2P. Puis, nous nous focalisons sur les systèmes de streaming P2P en dressant un panorama représentatif des systèmes existants à ce jour. Nous mettons l'accent sur le problème de disponibilité des données au sein de tels systèmes, en livrant une description formelle de cette problématique. Nous détaillons ensuite notre approche pour y remédier : la mise en cache des suffixes dans un cache virtuel distribué. Puis, nous présentons la stratégie adaptative que nous avons élaborée pour la gestion de ce cache. Les expérimentations que nous avons conduites pour tester la validité de notre démarche sont décrites et commentées en fin de chapitre.
4. **JStreaper - un système de streaming Pair-à-Pair :** ce chapitre présente les détails de la mise en œuvre du prototype JStreaper. Nous décrivons notamment son architecture logicielle ainsi que les considérations techniques retenues lors de l'implémentation : modes de communication, environnement de développement, etc.

   Ce manuscrit s'achève par une conclusion synthétisant nos contributions et mettant en relief les perspectives des recherches menées dans cette thèse. À noter que tous les sigles sont répertoriés dans un glossaire se trouvant à la fin du présent document.

Chapitre 1

Systèmes de streaming distribués : vue d'ensemble

Notre époque est incontestablement celle du « multimédia [1] ». En effet, les données statiques, de type texte ou image, ont perdu leur monopole au profit des données audiovisuelles, qui sont maintenant omniprésentes dans la vie quotidienne. Les brochures publicitaires se sont transformées en CDs multimédias. Les pannaux d'affichage se modernisent et deviennent progressivement des téléviseurs, capables de restituer des présentations animées, plutôt que de simples défilements d'images. Plus flagrant encore, les pages Web, qui contenaient majoritairement du texte et des images, regorgent actuellement de vidéos et d'animations. Dans ce manuscrit, nous nous focalisons tout particulièrement sur les données audiovisuelles.

Il existe deux moyens de consulter des contenus audiovisuels. Le premier est la restitution du contenu dont on dispose soit par la possession d'un support le contenant, soit par le téléchargement *via* un réseau. Ce moyen implique un temps d'attente non négligeable avant de disposer du contenu, sans parler du fait que l'on peut s'être trompé, pour une raison quelconque, et finalement s'être procuré un contenu qui ne nous intéresse guère. Le second moyen consiste à pouvoir visionner un contenu dès qu'on le souhaite, sans forcément le conserver par la suite. Le contenu est transmis *via* un réseau et est restitué aussitôt que les premières données arrivent. Cette méthode, appelée « streaming », n'implique pas, ou peu, de temps d'attente avant de pouvoir restituer le contenu, ce qui justifie sa popularité. L'intérêt suscité par les applications de streaming, comme la vidéo à la demande, a donné lieu à de nombreuses réalisations de systèmes permettant la mise en œuvre du streaming audiovisuel. Ces systèmes se sont fait appeler *systèmes de streaming* [WHZ+01].

L'apparition de terminaux capables de restituer des données audiovisuelles, combinée avec l'évolution des réseaux grand public, en terme de débit, a permis une accessibilité très large au streaming audiovisuel, ceci se faisant au travers de réseaux ayant des capacités (débit, latence, etc.) très variées, et en utilisant des terminaux ayant des capacités de traitement et de restitution très différentes. Les systèmes de streaming se sont vite trouvés limités et incapables de suivre la

1. De nos jours, le terme « média » est utilisé à tort et à travers. À l'origine, le mot media est le pluriel du mot latin *medium* qui signifie milieu ou intermédiaire (média de transmission, média de stockage, etc.). Dans ce manuscrit, par le mot *média* (ou *média de base*), nous entendons donnée. Les données (les médias) sont groupées et structurées dans des conteneurs appelés *objets médias*, *documents médias* ou tout simplement *contenus*.

demande, sans cesse croissante, de contenus audiovisuels de bonne qualité et en temps réel. Dès lors est venue l'idée de concevoir des *systèmes de streaming distribués* permettant de pallier ces limitations [DKSZ03].

Ce chapitre débute par une description des caractéristiques des données audiovisuelles qui imposent des contraintes strictes lorsqu'elles sont transmises en temps réel. Puis, nous poursuivons par une synthèse des différents concepts de base liés au streaming audiovisuel. La section 1.3, couvre les différentes facettes d'un système de streaming et les problématiques qui lui sont associées. Nous décrivons ensuite la notion de *cache* dans les systèmes de streaming ainsi que ses rôles, ses objectifs et les politiques de gestion existantes. Nous clôturons ce chapitre par une présentation de quelques systèmes de streaming distribués.

1.1 Les données audiovisuelles

On distingue deux classes de données : les données statiques ou discrètes de type texte ou image, et les données dynamiques ou continues de type audio ou vidéo. Les données dynamiques sont constituées d'une succession cohérente et ordonnée d'entités statiques ayant des dépendances temporelles entre elles. Ces entités peuvent être des échantillons dans le cas d'un son, ou bien des images dans le cas d'une vidéo.

1.1.1 Caractéristiques

À l'opposé des données statiques, les données audiovisuelles sont marquées par leur nature isochrone et extrêmement volumineuse.

1.1.1.1 Isochronisme

Une caractéristique contraignante et inhérente aux données dynamiques est leur dimension temporelle, appelée également contrainte temps réel ou contrainte de continuité. Ces données doivent être restituées à un rythme constant. Par exemple, une séquence vidéo n'est correctement visualisable que si les images la constituant sont affichées à une fréquence constante, en l'occurrence 25 images/seconde [2] (voir Figure 1.1). Cette dimension temporelle impose donc des contraintes strictes en matière de délais à respecter afin d'assurer la continuité des données. Ces contraintes se répartissent à tous les niveaux du cheminement des données audiovisuelles, depuis leur source jusqu'au point de restitution. À titre d'exemple, les unités de stockage doivent être capables de lui fournir les données avec un débit approprié.

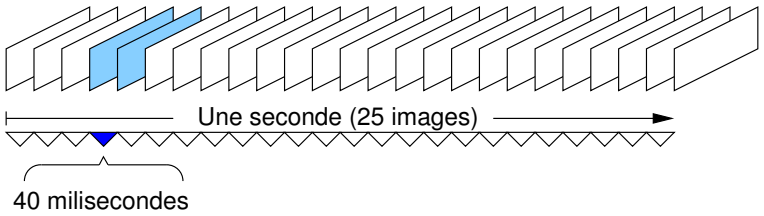


Figure 1.1 – Isochronisme dans une séquence vidéo.

1.1.1.2 Volume important

À l'inverse des données statiques, les donnée dynamiques sont extrêmement volumineuses et exigent une capacité de stockage très importante (voir Tableau 1.1). Des techniques de compression dédiées ont alors été développées afin d'en réduire le volume, tout en conservant la meilleure qualité possible d'écoute et de visualisation. Ces techniques permettent aux données audiovisuelles d'occuper moins d'espace de stockage. Toutefois, même après compression, les données audiovisuelles demeurent volumineuses. Par exemple, deux heures de vidéo, compressées selon le standard MPEG-2 avec un débit de 4 Mb/s, nécessitent approximativement un espace de stockage de 3,6 Go (cf. Tableau 1.2 [ATW02]).

2. La fréquence d'affichage est de 25 images/seconde pour les standards européens PAL et SECAM, alors qu'elle est de 30 images/seconde pour le standard américain NTSC.

Type de données	Volume
500 pages de texte ASCII (30 lignes, de 72 caractères par page)	1 Mo
1 heure de musique, CDA (stéréo, 44.1 kHz, 16 bits)	605 Mo
10 minutes de vidéo non compressées, PAL ($768X576$ pixels/frame, 24 bit/pixel)	149 Go

Tableau 1.1 – Volume des différents types de données.

Spécification de données		Domaine principal d'application	Débit
Données brutes (sans encodage)	Vidéo qualité TVDH $1920X1080$ pixels/frame 25 images/seconde, 24 bit/pixel	La télévision haute définition	1,16 Gb/s
	Vidéo qualité PAL $768X576$ pixels/frame 25 images/seconde, 24 bit/pixel	La télévision numérique	253 Mb/s
	Audio qualité DVD Surround, 96 kHz, 32 bits	Musique sur DVD	14,65 Mb/s
	Audio qualité CDA Stéréo, 44.1 kHz, 16 bits	Musique sur CD-ROM	1.35 Mb/s
Données encodées	Vidéo MPEG-2	La télévision numérique, DVD	3 à 20 Mb/s
	Vidéo MPEG-1	Vidéo sur CD-ROM	1.5 Mb/s
	Vidéo MPEG-4 (H.264)	Vidéo à la demande	10 à 100 Kb/s
	H.261	Visioconférence sur ISDN (p canaux)	$p \times 64$ Kb/s
	H.263	Visioconférence sur RTC	> 33 Kb/s
	Audio AAC	Musique sur PC ou baladeur	8 à 512 Kb/s
	Audio MP3	Musique sur PC ou baladeur	8 à 320 Kb/s

Tableau 1.2 – Principaux standards de codage audiovisuel.

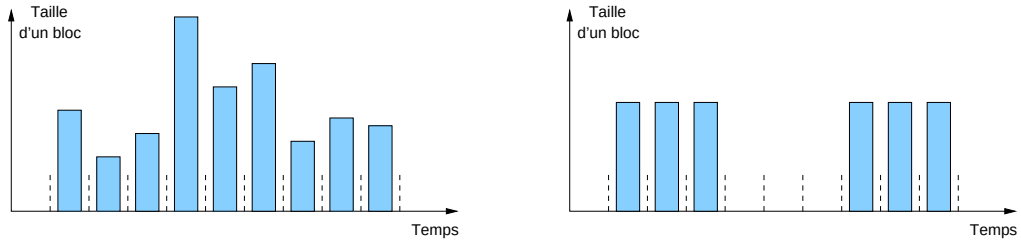
1.1.1.3 Compression irrégulière

Bien que chaque technique de compression de données audiovisuelles possède sa propre recette, elles sont toutes basées sur les mêmes principes. En effet, elles exploitent les limites de la perception humaine en éliminant tout ce qui est imperceptible par l'œil et/ou l'oreille humaine. Aussi, ces techniques s'appuient sur le principe de l'élimination des ressemblances dans des données contiguës. Ainsi, moins la vidéo contient de changements entre des images successives, plus les possibilités de compression sont importantes et *vice versa*.

Une vidéo peut contenir à la fois des séquences riches en mouvement et d'autres moins riches. Son taux de compression est donc amené à varier. Il en résulte une variabilité du débit lors de la lecture d'une séquence audiovisuelle compressée, mais sa qualité reste constante. Ces données sont appelées données à taux d'échantillonnage variable ou données VBR. Il existe d'autres méthodes de compression de contenus audiovisuels qui produisent des données à taux d'échantillonnage fixe au détriment de la qualité, qui devient variable. Ces données sont appelées données CBR.

Les données CBR sont stockées en blocs ayant la même taille et la même durée, tandis que pour les données VBR deux options sont envisageables (voir Figure 1.2) pour le stockage de données en blocs :

1. CTL (Constant Time Length) les blocs auront la même durée mais des tailles différentes,
2. CDL (Constant Data Length) les blocs auront la même taille mais des durées différentes.



(a) CTL (b) CDL

Figure 1.2 – Techniques de découpage de données audiovisuelles VBR.

Il est difficile de prendre en compte la variabilité du débit dans les données VBR. Des techniques de traitement ont donc été développées spécialement pour ces données. Néanmoins, dans leur majorité, les travaux de recherche dans le domaine de streaming audiovisuel font l'hypothèse que les données continues, après compression, ont un débit constant. On accepte une faible dégradation, souhaitée invisible, de la qualité. Ainsi, dans le reste de ce manuscrit, sauf mention explicite, nous nous focalisons sur des données audiovisuelles à débit constant.

1.1.2 Encodage

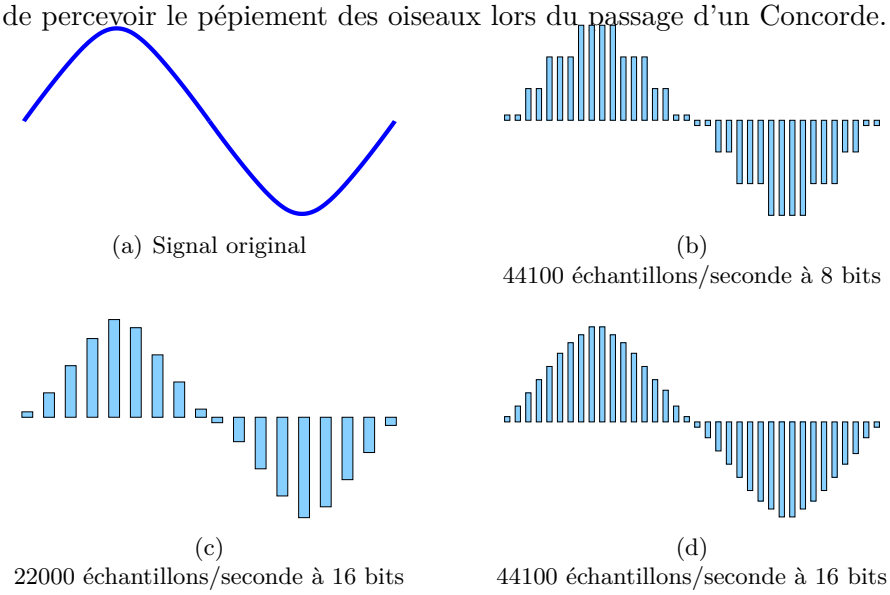
En raison de leur volume colossal, les données audiovisuelles sont compressées avant d'être emmagasinées (ou transmises), puis sont décompressées avant d'être restituées au spectateur. Les opérations de compression et de décompression, appelées aussi encodage et décodage, ont des caractéristiques opposées. En effet, si les algorithmes de compression peuvent être lents, complexes et coûteux, à l'exception de l'encodage en temps réel comme c'est le cas dans les visioconférence, il n'en va pas de même pour la décompression, qui est quant à elle simple et rapide. La compression ne s'effectue, généralement, qu'une fois au moment de la création, alors que la décompression s'effectue à chaque fois que le contenu est lu par un lecteur audiovisuel.

L'encodage/décodage de données audiovisuelles est effectué par ce qui est appelé *codec*. Les codecs peuvent être logiciels ou matériels, par le biais de cartes dédiées. D'ailleurs, ils sont souvent spécialisés pour l'encodage et/ou le décodage de données audio ou vidéo. On parle alors de codec audio et de codec vidéo. Néanmoins, les différents codecs, après avoir appliqué des algorithmes dédiés au son ou à la vidéo, font appel à des méthodes de compression dites *entropiques*

pour compresser davantage le flux audiovisuel résultant. Il s'agit de techniques de compression de données binaires sans tenir compte de leur significations comme l'encodage par plage (RLE), l'encodage de Huffman [Huf52, JJS93] ou plus récemment l'encodage arithmétique [WNC87, MW03]. À titre d'exemple, dans le codage par plage, toute suite de bits identique est remplacée par un couple (nombre d'occurrences ; suite répétée).

1.1.2.1 Encodage audio

Il existe plusieurs techniques de codage pour les contenus sonores, certaines sont dites *destructrices* et impliquent une perte de qualité, et d'autres non. Les techniques non destructrices exploitent les limites de perception de l'oreille humaine. Cette dernière entend les fréquences situées dans la gamme 20 Hz à 20 kHz. Si un morceau contient des fréquences hors de cette gamme, elles sont simplement supprimées. Une autre technique, considérée non destructrice, s'appuie sur le principe des fréquences masquées [Joh88]. En effet, si dans un groupe de fréquences, certaines ont un niveau sonore beaucoup plus élevé que d'autres, il n'est pas nécessaire de conserver les fréquences de niveau sonore faible : on ne les entendra pas. Par exemple, l'oreille humaine n'est pas capable de percevoir le pépiement des oiseaux lors du passage d'un Concorde.



(a) Signal original

(b) 44100 échantillons/seconde à 8 bits

(c) 22000 échantillons/seconde à 16 bits

(d) 44100 échantillons/seconde à 16 bits

Figure 1.3 – Son numérisé à différentes fréquences d'échantillonnage représenté pour différents nombres de bits.

Au rang des techniques destructrices, on trouve principalement les techniques qui permettent de modifier la qualité du son numérisé. La qualité du son est fonction de deux facteurs : la fréquence d'échantillonage et le nombre de bits utilisés pour représenter chaque échantillon. La Figure 1.3 illustre l'influence de ces deux facteurs. La qualité du son compressé est tributaire de la technique utilisée pour son encodage. Les techniques de compression s'étendent des algorithmes peu complexes qui produisent des taux de compression faibles et un son de mauvaise qualité comme la compression selon la loi-A, aux algorithmes complexes qui produisent des taux de compression élevés et un son de bonne qualité comme l'algorithme de compression MP3, en passant par les algorithmes d'une complexité moyenne qui produisent un son de qualité moyenne comme la compression différentielles adaptative (ADPCM).

La compression selon la loi-A [G.788], qui a fait l'objet d'un avis de normalisation de l'ITU-T, permet de réduire la qualité du son numérisé et résulte d'une fréquence d'échantillonage de 8 kHz codée sur 8 bits. Cette technique convient pour la compression de la parole et a été longtemps utilisée pour la téléphonie numérique ISDN. La technique de compression différentielle adaptative (ADPCM) tire profit du fait que les échantillons voisins sont semblables, et conserve uniquement la différence entre eux, améliorant par conséquent et la qualité du son compressé et le taux de compression [Pan93]. L'algorithme de compression spécifié par le standard MPEG-1, connu sous le nom MP3, exploite le principe de fréquences masquées et atteint des taux de compression nettement plus élevés par rapport aux techniques précédentes [Pan93]. Mieux encore, cet algorithme préserve la qualité du son perçu par l'oreille humaine. En effet, dans une série de tests, dont les détails se trouvent dans [GR91], des auditeurs experts n'ont pas pu faire la différence entre une bande son stéréo échantillonnée à 48 kHz et représentée par 16 bits, et sa version compressée avec cet algorithme à 256 Kb/s, soit un taux de compression de 6 :1.

Il existe d'autres techniques pour l'encodage audio telles que l'encodage hiérarchique [SJ98] ou l'encodage par descriptions multiples [AKG00]. Plus élaborées, ces techniques sont aussi utilisées pour l'encodage vidéo et sont décrites par la suite.

1.1.2.2 Encodage vidéo

Tout algorithme d'encodage/décodage vidéo exploite principalement les redondances résiduelles dans une séquence vidéo. Ces redondances sont à la fois spatiales, à l'intérieur d'une image, et temporelles, entre des images successives d'une vidéo. Nous distinguons trois méthodes d'encodage : encodage temporel, encodage hiérarchique et encodage par descriptions multiples.

Encodage temporel

L'encodage temporel se base sur deux principes : l'estimation et la compensation de mouvement entre les images successives d'une séquence vidéo [SS95]. Ces deux principes sont mis en œuvre respectivement par des opérations de prédiction et d'interpolation.

La prédiction se fait à partir d'images de référence dans une séquence vidéo. Ces images, baptisées images *Intra* (I), sont compressées de manière indépendante et contiennent toutes les données nécessaires à leur décodage. Les images *Prédites* (P), contiennent uniquement les différences par rapport à une autre image, qui peut être une image de référence ou une image prédite. L'interpolation est effectuée de manière bidirectionnelle en utilisant deux images, prédites ou de référence, produisant des images dénommées *Bi-prédites* (B). Ainsi, certaines images ne peuvent être reconstituées que si leur image de référence a été préalablement traitée. Une erreur produite lors d'un traitement, ou d'une transmission, d'une image de référence est donc propagée aux images prédites et interpolées suivantes. La figure 1.4 illustre l'ordre de décodage de différents types d'images selon le standard MPEG-1.

Notons que toutes les images, après encodage temporel, sont encodées spatialement par transformation en cosinus discrète [Kha03]. Le but est de représenter les données dans un autre espace, en l'occurrence celui des fréquences spatiales. Les images sont ensuite quantifiées. Il s'agit de la seule étape de la chaîne de compression qui occasionne des pertes de données (par exemple : unification de plages de couleurs comportant des variations relativement petites).

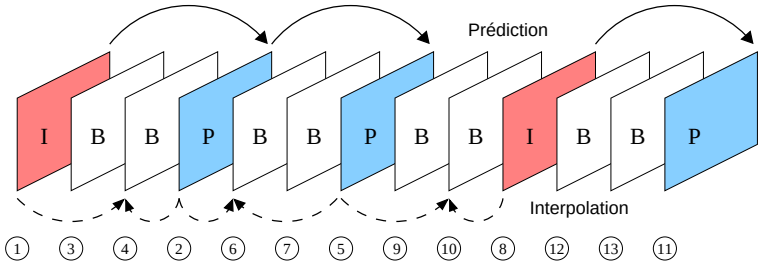


Figure 1.4 – Ordre de codage dans le modèle IPB de la norme MPEG-1.

La quantification peut donc ne pas introduire de perte de qualité en tirant profit des limites de perception de l'œil humain. Toutefois, afin d'atteindre un taux de compression élevé, un sacrifice sur la qualité peut s'avérer nécessaire.

Encodage hiérarchique

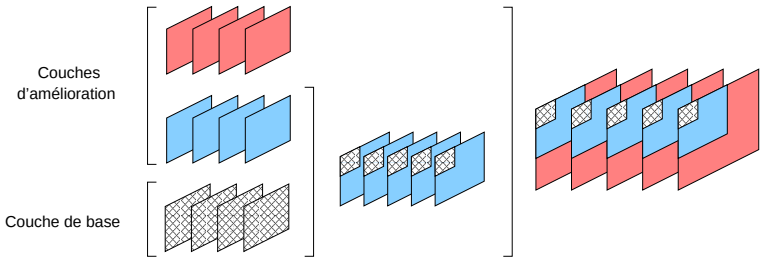
L'encodage hiérarchique d'une séquence vidéo en multi-couche (layered compression), appelé aussi *compression échelonnée* ou *encodage scalable*, se traduit par une décomposition en ondelettes [TZ94, FG04]. Il consiste à compresser un contenu en plusieurs couches correspondant à différents niveaux de qualité, une couche de base (« base layer », BL) et une ou plusieurs couches d'amélioration (« enhancement layer », EL) que l'on peut utiliser l'une après l'autre pour raffiner la qualité de la couche de base. Notons que l'amélioration de la qualité peut prendre des dimensions spatiales [WWL$^+$01] ou temporelles [WLZ01, SR01], voire une combinaison des deux [GFTK06] (cf. Figure 1.5).

L'encodage hiérarchique est incontestablement meilleur que l'encodage temporel en termes de flexibilité, d'efficacité et surtout de scalabilité [SWS03]. En effet, en cas de besoin, on peut abandonner des couches d'amélioration, en commençant par la dernière, donnant un contenu de moindre taille nécessitant moins de débit, mais aussi de moindre qualité. Son seul inconvénient est qu'il nécessite plus de puissance de calcul que l'encodage temporel et atteint des taux de compression à peine moins élevés à cause des données supplémentaires nécessaires à la gestion de la hiérarchie.

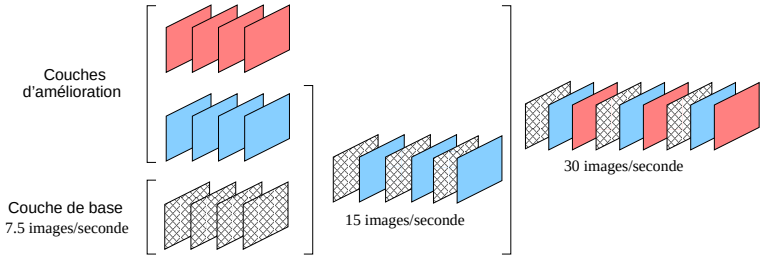
Encodage par description multiple

L'encodage par descriptions multiples (MDC), comme nous allons le voir plus tard dans ce chapitre, est une technique très robuste pour la transmission de contenus audiovisuels en temps réel [Goy01]. Cet encodage se traduit par la construction de plusieurs descriptions indépendantes pouvant contribuer à la restitution spatiale ou temporelle d'un contenu dynamique. Ces descriptions peuvent avoir la même importance ou des importances différentes, on parle respectivement d'encodage MDC équilibré et d'encodage MDC déséquilibré.

De même que pour l'encodage hiérarchique, on peut abandonner des descriptions en cas de besoin pour avoir un contenu dont la qualité est dégradée mais dont la taille et le débit sont réduits. La différence par rapport à l'encodage hiérarchique réside dans la dépendance. En effet, à l'inverse des descriptions qui sont indépendantes les unes des autres, les couches dans l'encodage hiérarchique sont utilisées l'une après l'autre pour affiner la qualité.



(a) Hiérarchisation spatiale



(b) Hiérarchisation temporelle

Figure 1.5 – Hiérarchisation spatiale et temporelle.

1.1.3 Formats audiovisuels

Les codecs compressent les données audiovisuelles dans des formats précisés par des standards. Ceux-ci peuvent être libres tels que H.261/3/4 [H.293, H.297, SWS03] de l'ITU-T ou MPEG-1/2/4 [JTC93, JTC99, JTC00] de l'ISO ou bien propriétaires comme les solutions proposées par RealNetworks®, Apple® ou Microsoft®. Les différents formats peuvent correspondre à différents besoins en qualité de restitution, de temps de compression ou de décompression, de débit du flux après compression ou de taille du contenu compressé résultant. Voici un bref descriptif de quelques uns des formats audiovisuels les plus utilisés :

- **MPEG** : ces formats sont définis par les normes MPEG-1/2/4. En particulier, la couche audio du standard MPEG-1, souvent référencée comme la 3^{e} couche, MP3, est très populaire grâce à son taux de compression très intéressant.

- **AAC** : ce format, devenu maintenant un standard, est le successeur de MP3. En effet, il offre un meilleur rapport qualité/compression que le format MP3.

- **DivX®** : ce format, développé par DivX® Inc., est à la vidéo ce que le MP3 est à la musique. Il est l'implémentation la plus connue du standard MPEG-4. Il permet d'avoir des fichiers de taille très réduite avec une qualité d'image supérieure à celle obtenue avec beaucoup d'autres formats.

- **QuickTime®** : ce format, développé par Apple® Computer Inc., est devenu un standard de fait pour les éditeurs professionnels de contenus audiovisuels.

- **Real® Media** : ces formats, créés par RealNetworks®, sont parmi les premiers formats conçus spécialement pour la diffusion de contenus audiovisuels en temps réel.

- **Flash® Media** : ces formats sont actuellement développés par Adobe® Systems Inc. qui a acquis leur créateur Macromedia® Inc. L'utilisation de ces formats est en forte croissance pour la transmission de séquences audiovisuelles en temps réel sur le Web.

- **Windows® Media** : ces formats sont développés par Microsoft® Corporation pour être utilisés uniquement avec son lecteur Windows® Media Player.

- **Theora et Vorbis** : ces formats, créés par la fondation Xiph, existent à la fois pour la vidéo (Theora) et pour le son (Vorbis). Ils sont développés de façon totalement libre et gratuite, et ne font l'objet d'aucun brevet logiciel.

Aucun des ces formats n'est vraiment parvenu à s'assurer l'hégémonie sur le marché de l'audiovisuel. La plupart des éditeurs de contenus audiovisuels sont donc obligés de proposer leurs contenus sous au moins deux formats, et ce dans le but de toucher le plus de spectateurs possible. Tous ces formats devraient donc cohabiter encore un certain temps.

Les éditeurs de contenus audiovisuels privilégient souvent l'un ou l'autre de ces formats, en fonction de son domaine d'application. Lorsque les données continues sont destinées à être transmises en temps réel, le format adopté est souvent tributaire du débit de connexion des usagers [3]. Salué pour l'excellente qualité de ses images, QuickTime® est privilégié pour la diffusion vidéo à haut débit. Sur le segment des connexions à faible et moyen débit, les formats de RealNetworks® et Microsoft® semblent être mieux adaptés.

1.2 Streaming audiovisuel

Stream est un mot anglais signifiant flux. Tout type de données, même textuelles, peut être transmis en flux continu. Ceci permet de les rendre disponibles pour les restituer aussitôt qu'elles arrivent et éviter ainsi l'attente du téléchargement de l'ensemble des données. Le terme *streaming* peut être défini comme l'ensemble des technologies permettant le téléchargement et la restitution en simultané de données *via* un réseau. Dans ce manuscrit, nous nous focalisons sur les données audiovisuelles. Nous appelons stream, un flux de données audiovisuelles et nous entendons par streaming, la « transmission en continu » de données audiovisuelles.

Les contraintes temps réel inhérentes aux données audiovisuelles font que la mise en œuvre du streaming n'est pas une tâche aisée. Cette section introduit le streaming en le comparant avec d'autres modèles de transmission. Les avantages, les modes et les applications du streaming sont également présentés.

1.2.1 Streaming versus téléchargement

Les données dynamiques peuvent être affichées à partir d'une machine locale ou d'une machine distante. Dans le premier cas, les données sont entièrement disponibles localement, stockées sur un disque dur ou disponible sur un DVD par exemple. Lorsqu'un spectateur demande la restitution d'un contenu audiovisuel, les données sont récupérées depuis l'espace de stockage local, décodées, puis passées à la carte graphique et/ou carte son qui se charge(nt) de les lui restituer.

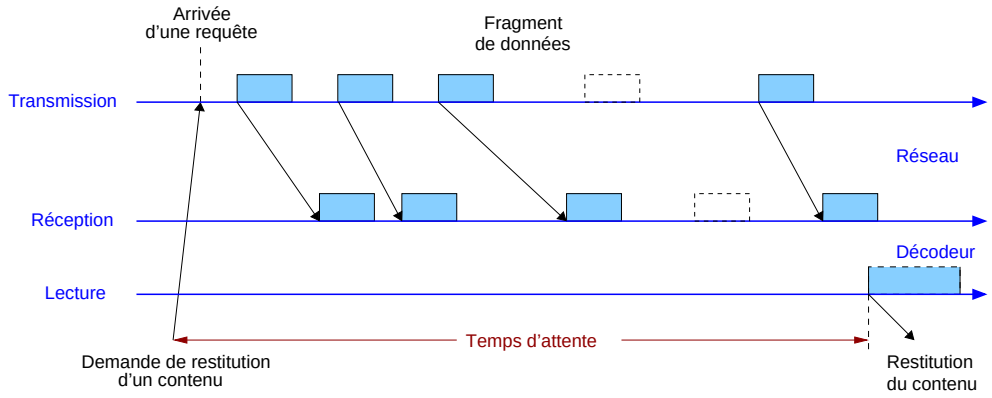
3. Les termes : spectateur, auditeur, utilisateur, et usager sont utilisés indifféremment dans le reste de ce manuscrit.

Lorsqu'un spectateur demande la restitution d'un contenu audiovisuel se trouvant sur une machine distante, les données sont récupérées depuis l'espace de stockage de cette machine, puis sont transmises à la machine de l'utilisateur par l'intermédiaire d'un réseau. Trois modèles de transmission sont possibles : téléchargement simple, streaming, ou téléchargement progressif.

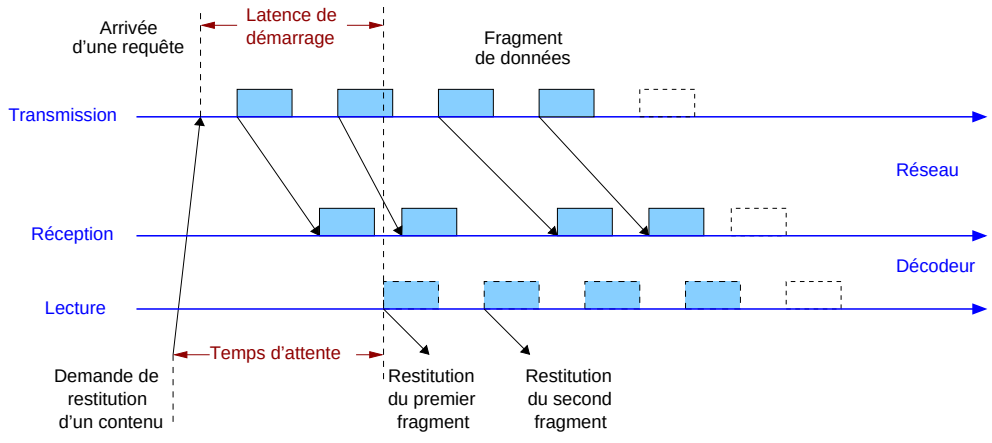
1.2.1.1 Téléchargement simple

Dans ce modèle, appelé aussi *emmagasiner puis restituer*, les données sont entièrement téléchargées d'une machine distante vers un espace de stockage local avant de commencer leur décodage puis leur restitution (voir Figure 1.6). Le transfert de données peut se faire à l'aide d'un serveur HTTP ou FTP standard. Le temps de transfert augmente avec le nombre de requêtes à servir simultanément. Il est fonction de la taille du contenu à télécharger et du débit de transfert :

$$\text{Temps de transfert (s)} = \frac{\text{Taille du contenu (Mb)}}{\text{Débit de transfert (Mb/s)}}$$



(a) Téléchargement simple



(b) Streaming

Figure 1.6 – Téléchargement versus streaming

1.2.1.2 Streaming

Ce modèle [DKSZ03, ATW02, SD00], appelé aussi *restitution instantanée*, permet la lecture d'un flux peu après la réception des premières données, sans attendre le téléchargement complet du contenu. Les données sont récupérées depuis une machine distante par l'intermédiaire d'un réseau, puis décodées et restituées à l'utilisateur dès qu'elles arrivent à sa machine (voir Figure 1.6). Les données ne sont pas enregistrées dans l'espace de stockage local. Toutefois, elles peuvent être mises en mémoire tampon avant d'être décodées (« bufferisation ») .

Cette technique n'est envisageable que si la machine émettrice ainsi que le réseau peuvent garantir les contraintes de continuité des données dynamiques. Le streaming est très sensible aux délais de propagation. Suivant l'application considérée, les données perdues ou qui ne sont pas arrivées dans un certain délai (de l'ordre de quelques dizaines de millisecondes) deviennent inutiles. La restitution des données risque par conséquent d'être perturbée. Une autre condition contraignante de l'utilisation du streaming est que les données dynamiques doivent être encodées de manière à ce que les fragments soient décodables indépendamment les uns des autres.

D'une manière formelle, le streaming peut être exprimé comme une suite de contraintes. Supposons que les fragments de données doivent être restitués à des intervalles de Δ secondes. Chaque fragment doit être délivré et décodé avant l'instant de sa restitution :

- le fragment N doit être délivré et décodé avant le moment de sa restitution : T_N ;
- le fragment $N + 1$ doit être délivré et décodé avant $T_N + \Delta$;
- le fragment $N + 2$ doit être délivré et décodé avant $T_N + 2\Delta$;
- etc.

1.2.1.3 Téléchargement progressif

Ce modèle, appelé également *pseudo streaming*, permet de commencer la lecture avant l'arrivée complète du contenu. L'affichage commence dès qu'une quantité suffisante de données est localement disponible. Le reste des données continue à être récupéré et emmagasiné dans l'espace de stockage local. Au final, les données seront entièrement stockées dans l'espace de stockage local. Le téléchargement progressif permet ainsi au client de lire le début du contenu pendant que le reste se télécharge et aussi de conserver une copie locale du contenu. Le modèle peut être considéré comme un cas spécial du téléchargement simple ou alors un cas spécial du streaming.

1.2.2 Avantages

La transmission de données audiovisuelles en streaming présente plusieurs avantages :

- Il permet aux usagers de commencer à visualiser, ou entendre, un contenu audiovisuel, volumineux par nature, sans attendre que la totalité des données soit téléchargée.
- Il offre la possibilité au spectateur de se rétracter à tout moment, si le contenu ne correspondait pas à ce qu'il attendait.
- Il ne nécessite pas de disposer d'un espace de stockage chez l'utilisateur. Toutefois, un espace limité de mémoire tampon peut être demandé pour le décodage des données.
- Aucune copie locale n'est conservée. Ceci est très utile pour le partage de contenus soumis à des licences restrictives quant à la distribution de contenus (mais qui en autorise la diffusion).
- Il est bien adapté pour la transmission des événements en direct.

1.2.3 Modes

En fonction de l'efficacité de transmission, on peut distinguer deux modes de transmission de données audiovisuelles : la diffusion individuelle et la multidiffusion.

1.2.3.1 Diffusion individuelle

Dans ce mode, appelé aussi transmission *point à point* ou *un vers un*, un émetteur envoie un flux de données audiovisuelles par spectateur, il y a autant de flux que d'auditeurs (voir Figure 1.7). La diffusion individuelle (« unicast », « one-to-one ») nécessite donc des émetteurs puissants et dotés d'une bande passante de sortie très élevée. Un canal de retour (« feedback ») est souvent associé au canal d'émission donnant la possibilité à l'émetteur de modifier son comportement en fonction de l'état du récepteur ou du canal. Ce mode de diffusion permet au récepteur de contrôler le flux de données en le mettant en pause par exemple. La vidéo à la demande est une application très populaire utilisant ce mode de transmission.

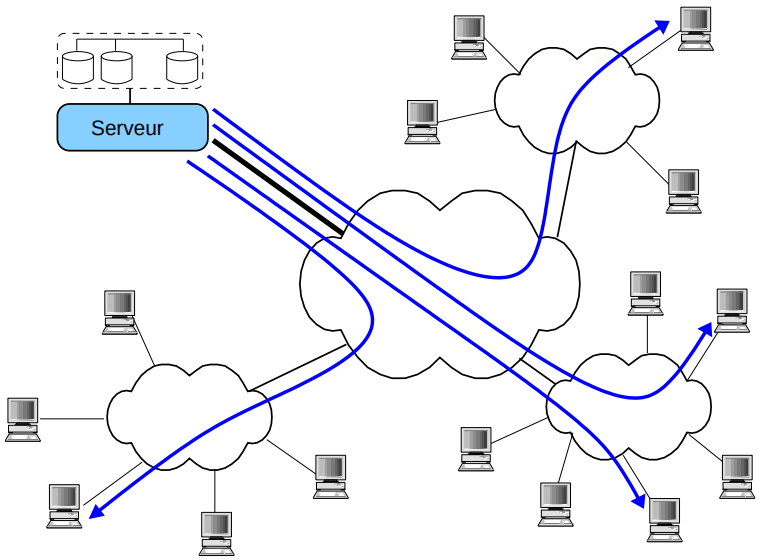


Figure 1.7 – Diffusion individuelle de vidéos à la demande.

1.2.3.2 Multidiffusion

La multidifusion, appelée aussi transmission *multipoints* ou *un vers plusieurs*, est certainement plus efficace pour la distribution de contenus audiovisuels à un très grand nombre de récepteurs, sous réserve qu'elle soit supportée par le réseau sous-jacent. Par un mécanisme d'abonnement, les spectateurs peuvent s'enregistrer dans un groupe de multidiffusion. Un émetteur, appartenant au groupe, peut ainsi envoyer, en une fois, un flux des données audiovisuelles à l'ensemble des récepteurs enregistrés dans le groupe (cf. Figure 1.8). Ce mode augmente donc la capacité de l'émetteur à servir simultanément plus de spectateurs, cependant le retour d'informations d'un récepteur vers l'émetteur est généralement infaisable limitant ainsi la capacité de l'émetteur à s'adapter aux récepteurs. Le spectateur ne peut donc pas piloter le flux ; il doit

suivre le programme imposé comme pour une télédiffusion. La radio en ligne ou la télévision en ligne sont des exemples courants d'applications utilisant ce mode de streaming.

Un cas particulier de ce mode est la diffusion large (« broadcast », « one-to-all »), où un émetteur envoie un flux de données audiovisuelles à tous les spectateurs connectés.

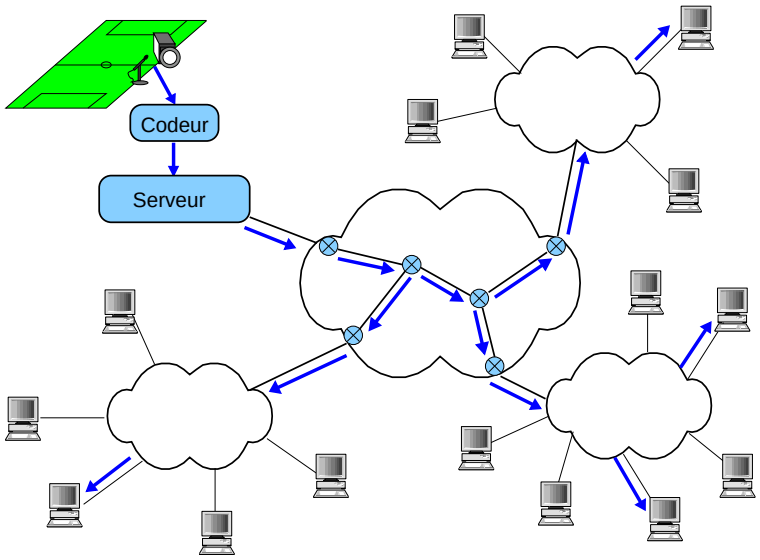


Figure 1.8 – Multidiffusion d'un événement en direct.

Bien que la multidiffusion soit de plus en plus utilisée, elle n'est pas encore déployée à large échelle sur Internet. Ceci est la conséquence directe du fait que les communications sur Internet se font aujourd'hui en utilisant principalement des transmissions *point à point*.

1.2.4 Applications

Les diverses applications de streaming s'étendent à un public très large et dispersé géographiquement. Dans cette section, nous présentons, dans l'ordre de leur complexité croissante, une taxinomie d'applications qui échangent des flux en streaming en mettant en relief leur caractéristiques.

1.2.4.1 Diffusion en direct

Les données audiovisuelles peuvent être capturées et encodées en temps réel pour être diffusée, tout de suite, en direct (« Live »). Nous appelons le stream résultant *stream vivant*, alors que nous appelons le résultat d'une transmission en streaming des données audiovisuelles préalablement encodées et stockées, *stream rémanent*.

Lorsqu'un émetteur diffuse un stream vivant, issu d'un événement en direct par exemple, de nombreux utilisateurs se connectent au stream afin de le regarder ou de l'écouter (voir Figure 1.8). Ce stream, dont la durée peut être, ou non, connue à l'avance, est transmis de la même manière à tous les spectateurs qui ne peuvent pas le contrôler. On parle d'une participation passive des spectateurs.

1.2.4.2 Diffusion à la demande

Dans ce type d'applications, un utilisateur émet une requête qui est traitée uniquement pour lui (voir Figure 1.7). Le stream rémanent est délivré en différé vers les spectateurs qui en ont fait la demande. Les usagers de ce type d'applications tolèrent, en général, un temps de latence initial avant le démarrage du flux demandé. La durée des streams est toujours connue à l'avance et peut être longue (films de cinéma) ou relativement courte (nouvelles, publicités, etc.). Ils peuvent être structurés ou indépendants les uns des autres. Par streams indépendants, nous entendons qu'ils ne possèdent pas de liens structurels et sémantiques les reliant.

L'exemple le plus courant de ce type d'applications est le média à la demande [4] (« Media-On-Demand« , MOD) qui se décline en trois catégories selon le degré de l'interactivité permise à l'usager [LV94, SD00] :

True Media-On-Demand (T-MOD) : Un système MOD est dit T-MOD, s'il permet aux utilisateurs de a) accéder à n'importe quel stream disponible dans le système, b) ceci à tout moment, et c) effectuer un contrôle complet sur les streams y compris les fonctions de magnétoscope classiques telles que l'avance rapide et le recul rapide.

Near Media-On-Demand (N-MOD) : Un système MOD est dit N-MOD, si au moins une des conditions citées pour le T-MOD est violée. Communément, les conditions b) et c) ne sont pas validées par la majorité des N-MOD existants. Ces systèmes diffusent certains streams périodiquement (toutes les 5 minutes par exemple) en interdisant aux utilisateurs de les contrôler. L'inconvénient majeur de ce système, par rapport à un système T-MOD, est que les requêtes arrivant tôt doivent attendre les retardataires. Ceci entraîne la non-équité entre les requêtes en terme de temps d'attente et conduit donc à une probabilité élevée de rétractation chez les utilisateurs.

Quasi Media-On-Demand (Q-MOD) : Un système Q-MOD est un système N-MOD basé sur des seuils. Au lieu de diffuser les streams périodiquement, un stream est diffusé uniquement lorsqu'il y a au moins k requêtes à ce stream. Un utilisateur demandant un stream doit attendre que le nombre minimal de requêtes demandant le même stream soit obtenu sur le serveur Q-MOD. L'équité entre les requêtes est fonction du seuil k. Elle croit avec la fréquence d'arrivée des requêtes désirant un stream donné.

1.2.4.3 Visioconférence

Dans les applications examinées ci-dessus, les utilisateurs sont isolés, sans interactions entre eux. Dans une application de visioconférence (ou de téléphonie par Internet), les utilisateurs deviennent des interlocuteurs ; ils se parlent et se voient en temps réel par le biais de streams vivants. En plus des contraintes temps réel inhérentes aux données audiovisuelles, ce type d'applications interactives nécessite des temps de réponse très courts, de l'ordre de 150ms [ATW02].

La visioconférence permet de transformer les réunions physiques en réunions virtuelles réduisant le temps perdu et les coûts de voyage. D'ailleurs, on observe une demande croissante d'environnements, fonctionnels et peu onéreux, pour les téléréunions internationales, les téléformations à distance, etc. L'Auditorium [Che01] est un exemple d'un système de visioconférence, basé sur la technologie Java.

4. La vidéo à la demande (VOD) est un cas particulier.

1.3 Système de streaming

Un *système de streaming* , appelé aussi *environnement de streaming*, se décompose en trois parties : le serveur chargé de « streamer » les données audiovisuelles, c'est-à-dire de les transmettre en flux continu ; les lecteurs qui restituent ces données aux utilisateurs ; et le réseau les connectant (voir Figure 1.9). Chacune de ces parties doit satisfaire à certaines conditions requises pour que le streaming soit faisable. Dans cette section, nous examinons, plus ou moins en détails, chacune de ces trois parties. Nous décrivons également les deux catégories de systèmes de streaming, en l'occurrence « client pull » et « serveur push ».

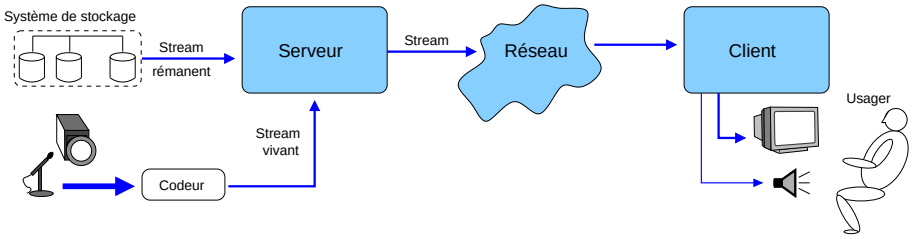


Figure 1.9 – Système de streaming.

1.3.1 Lecteur audiovisuel

Dans un système de streaming, la partie qui est en contact direct avec les utilisateurs est le lecteur audiovisuel. Dans la terminologie du génie logiciel, le lecteur audiovisuel est appelé *client*. Le client est un logiciel permettant de lire des données audiovisuelles ; il permet de les décoder et de les restituer. En plus, il comporte une interface utilisateur qui prend en compte certains désirs des usagers par le biais d'un certain nombre d'interactions. Dans la suite, nous introduisons les trois composants principaux d'un client : le décodeur, le démultiplexeur et le contrôleur. Un panorama des lecteurs audiovisuels les plus populaires est ensuite présenté.

1.3.1.1 Décodeur

Comme cela a été mentionné dans la Section 1.1, en raison de leur nature extrêmement volumineuse, les données audiovisuelles sont emmagasinées et transmises sous forme compressée. Le décodeur est le composant chargé de décompresser (décoder) les données avant leur restitution. Un décodeur est souvent spécialisé dans la décompression de données audio ou vidéo. On parle alors de décodeur audio et décodeur de vidéo. Plus les décodeurs disposent d'algorithmes de décodage, plus le lecteur est capable de lire de formats de données différents.

Le décodage de données audiovisuelles doit être très rapide afin de respecter leurs contraintes temporelles intrinsèques. Il nécessite donc une puissance de calcul relativement élevée. D'ailleurs, la puissance requise pour décoder un contenu donné dépend de son format. À titre d'exemple, le format H.264 est plus compliqué que MPEG-2 et requiert plus de puissance de calcul pour être décodé tout en respectant l'isochronisme des données. Ainsi, la machine hébergeant un lecteur audiovisuel doit être dotée d'une puissance de calcul suffisante pour décoder les différents formats de données audiovisuelles. Cette contrainte semble être dépassée par l'avancée technologique que l'on vit actuellement. En effet, même les terminaux mobiles possèdent aujourd'hui une capacité

de traitement suffisante pour décoder les contenus audiovisuels qui leur sont destinés : des contenus de petite résolution pour être affichés sur des écrans de petites dimensions.

1.3.1.2 Démultiplexeur

Les données audiovisuelles, après encodage, se trouvent souvent groupées dans des conteneurs, appelés aussi « containers » ou « wrappers », afin de faciliter leur stockage et/ou leur transmission. On parle alors de *données multiplexées*. Certain conteneurs, tels que Matroska ou Ogg, peuvent inclure la vidéo d'un film, plusieurs pistes audio et plusieurs pistes de sous-titres. Ce groupement, ou *multiplexage*, de données compressées, est effectué par des logiciels appelés *multiplexeurs*, alors que le dégroupement (*démultiplexage*) ou l'extraction de différents flux, toujours compressés, depuis des données multiplexées est effectué par des *démultiplexeurs*. Le démultiplexeur est ainsi le composant chargé d'extraire les différents flux de données multiplexées.

1.3.1.3 Contrôleur

Le rôle principal du contrôleur est de gérer les interactions des utilisateurs, mais aussi de superviser le cheminement des données audiovisuelles. Les différentes commandes des utilisateurs sont déléguées au système d'exploitation sous-jacent (OS), chargé de les exécuter, par le biais d'une interface (API) qui lui est associée [RS93]. Le chemin qu'empruntent les données audiovisuelles comprend, quant à lui, trois phases. Les données compressées et multiplexées sont d'abord récupérées depuis une source jusqu'au démultiplexeur qui extrait les données audio et vidéo compressées. Les données sont ensuite décompressées par les décodeurs appropriés. Enfin, les données décodées sont acheminées jusqu'au point de restitution (voir Figure 1.10).

Les interactions classiquement offertes aux utilisateurs par un lecteur audiovisuel sont l'augmentation ou la diminution du volume sonore, la mise en pause, l'arrêt, la reprise, l'avance rapide, le recul rapide et la navigation temporelle dans un contenu audiovisuel. Un client doit également fournir une interface permettant aux usagers de sélectionner les contenus qui les intéressent.

Les informations apportées aux spectateurs concernant un contenu donné varient selon l'application. Ces informations peuvent être le format, une brève description, le prix à payer pour un contenu donné, etc. L'ensemble des interactions potentiellement fournies par un client donné sont, elles aussi, spécifiques à l'application. Par exemple, dans une application de diffusion en direct, les interactions possibles se réduisent à l'arrêt, la reprise et la variation du volume du contenu audiovisuel. En revanche, un client conçu pour une application de visioconférence offre d'autres fonctionnalités, par exemple demander la permission de parler. Un autre exemple, plus flagrant, est constitué par les fonctionnalités d'avance rapide et de recul rapide, qui ne sont prises en charge actuellement dans aucune application de streaming ! En effet, c'est une problématique qui suscite l'intérêt de nombreux chercheurs dans le monde entier. Elle est évoquée plus amplement dans le Chapitre 2.

1.3.1.4 Clients existants

Il existe de nombreux clients permettant la lecture des données audiovisuelles. Plutôt qu'une énumération exhaustive mais sommaire, nous en décrivons quelques uns parmi les lecteurs les plus populaires.

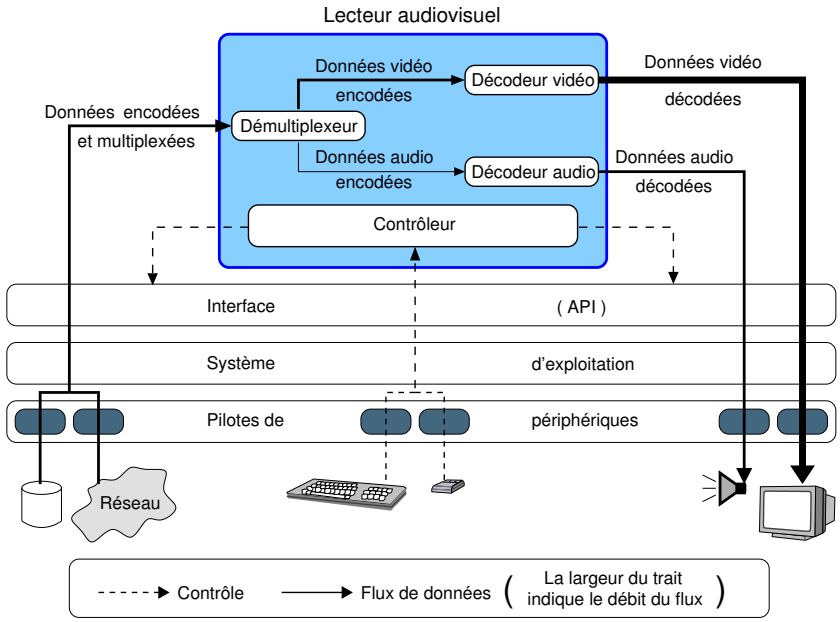


Figure 1.10 – Composants d'un lecteur audiovisuel.

VLC media player : Le lecteur VLC est la partie cliente du projet VideoLAN, qui constitue une solution complète pour la lecture et la diffusion de données audiovisuelles. Un des grands atouts de VLC est qu'il intègre les codecs nécessaires à la lecture de la plupart des formats audio et vidéo. Il est disponible sur la plupart des plates-formes. À l'origine, il était développé par les étudiants de l'École Centrale Paris et a été diffusé pour la première fois le 1er février 2001 sous licence GPL. Il est aujourd'hui développé par des contributeurs du monde entier. Le projet reste toutefois coordonné par des élèves de deuxième année et plus de l'École Centrale Paris.

MPlayer : C'est un lecteur audiovisuel libre et distribué sous licence GPL, porté sur plusieurs plates-formes telles que Mac OS® X, Microsoft® Windows® et Solaris™. Il est connu pour prendre en charge un très grand nombre de formats vidéo. Il est accompagné de MEncoder, qui est à la fois un outil d'encodage (ou transcodage) et de montage audio et vidéo.

Windows® Media Player : C'est un lecteur audiovisuel propriétaire produit par l'entreprise Microsoft®. Il est essentiellement disponible pour les systèmes d'exploitation Microsoft® Windows®. Selon une étude de Nielsen [Nie07], il occupe la première place du marché mondial du streaming.

RealPlayer® : C'est un lecteur audiovisuel édité par RealNetworks®. Il est très semblable à Windows Media Player en termes de rapidité, de qualité de son et de qualité de l'image lors de la lecture de vidéos ou de DVD. Il fonctionne grâce à un moteur à sources ouvertes (« open source ») appelé Helix™. Il existe des versions « basiques » gratuites de ce client ainsi que des versions payantes avec des fonctionnalités supplémentaires. Il est également disponible sur la plupart des plates-formes.

QuickTime® et iTunes® : Ce sont des lecteurs distribués gratuitement par Apple®. Ils sont disponibles officiellement sur Mac OS® X, Microsoft® Windows®, et officieusement sur les systèmes GNU/Linux. Selon la même étude de Nielsen [Nie07], ils occupent, en tant que logiciels de streaming, la 3e place du marché derrière Windows® Media Player et RealPlayer®.

Adobe® Flash® Player : Il est officiellement compatible avec les systèmes d'exploitation Windows®, GNU/Linux, Mac OS® X, Solaris™, mais sa compatibilité GNU/Linux se résume aux architectures x86 32 bits. Aujourd'hui sa popularité se voit en stricte croissance grâce à son plug-in disponible pour la majorité des navigateurs Web, car ce plugin est fortement utilisé par les sites Web consacrés au partage de vidéo en streaming comme Youtube ou Dailymotion.

1.3.2 Réseaux et streaming

Dans un système de streaming, le terme réseau englobe les infrastructures matérielles et les protocoles logiciels permettant de transmettre au serveur les commandes des utilisateurs et de délivrer aux clients les données audiovisuelles choisies, tout en respectant leur isochronisme. Or, la transmission sans garantie, offerte par les réseaux à commutation de paquets, a des effets néfastes sur la continuité des données audiovisuelles. Pour contrer ces effets, deux voies sont envisageables : les applications ajustent leur comportement en fonction de l'état du réseau, et/ou le réseau s'adapte au transfert des données dynamiques.

Au niveau applicatif, on trouve dans la littérature des techniques comme le streaming adaptatif ou bien la correction des erreurs de transmission. Ces techniques sont examinées dans la Section 1.3.3.6. Au niveau réseau, un ensemble de protocoles répondant aux besoins temps réel des données audiovisuelles a été proposé. Dans cette section, nous décrivons d'abord les problématiques liées au streaming sur des réseaux à commutation de paquets, en particulier les réseaux basés sur le protocole IP. Puis, nous présentons les différents protocoles conçus spécialement pour le streaming de données audiovisuelles.

1.3.2.1 Problématiques dues aux réseaux

Dans les réseaux à commutation de paquets, comme l'est actuellement Internet, le délai de transmission d'un paquet n'est pas garanti. D'ailleurs, ce délai présente des fluctuations importantes. La variation du délai de transmission est communément appelée *gigue*. La gigue a des effets perturbateurs voire destructeurs des relations temporelles intrinsèques aux données dynamiques, causant ainsi la *désynchronisation* de celles-ci. Elle peut empêcher la reconstitution correcte des données dynamiques qui doivent être décodées et restituées à une cadence constante. Afin de réduire l'effet de la gigue, un tampon de lecture est souvent utilisé du côté client. Cette technique, située au niveau applicatif, est décrite plus amplement dans la Section 1.4.

L'arrivée des paquets n'est pas garantie non plus dans les réseaux à commutation de paquets. En outre, les données peuvent être altérées lors de la transmission. La perte ou la transformation des données a des effets nuisibles à la qualité des données reconstituées. Au vu des contraintes temporelles inhérentes aux données dynamiques, les paquets perdus ne sont retransmis que s'ils peuvent être délivrés avant qu'il soit temps de les décoder pour être restitués au spectateur. Néanmoins, des paquets perdus ou transformés peuvent être reconstitués par des mécanismes de contrôle d'erreur qui sont décrits dans la Section 1.3.3.6.

Aussi, les réseaux à commutation de paquets n'offrent pas la possibilité de réserver une bande passante pour le transfert des données audiovisuelles, volumineuses par nature. De plus, l'hétérogénéité des infrastructures réseaux, utilisant des technologies très variées, implique de grandes différences quant à leurs capacités en bande passante. Ainsi, la bande passante entre deux points sur Internet est en général inconnue et variable dans le temps. Si un émetteur envoie

à un débit supérieur au débit supporté par le réseau, il y a congestion, ce qui implique que des paquets sont perdus et donc la qualité des streams diminue considérablement. En revanche, si la transmission se fait à un débit très en-dessous de la bande disponible, le stream transmis est sous-optimal. Afin de remédier à ce problème, on adapte le taux de transmission de données au débit de transmission dicté par les conditions du réseau, on parle alors de *streaming adaptatif*.

Des difficultés supplémentaires apparaissent lorsque les données traversent des réseaux sans-fil caractérisés par une bande passante réduite, des délais considérablement plus élevés, des taux d'erreurs plus élevés, et des problèmes de connexion intermittente. Les streams doivent donc être adaptés à ces environnements mobiles. Cette adaptation se traduit par une diminution du débit des streams.

1.3.2.2 Protocoles dédiés au streaming

Afin de masquer la complexité et l'hétérogénéité des infrastructures réseaux aux développeurs d'applications et aux utilisateurs finaux, les différents aspects de la communication sont modélisés en plusieurs couches. En principe, une couche ne peut communiquer qu'avec les couches de rang immédiatement supérieur et/ou inférieur. La modélisation en couches permet de modifier et même remplacer une couche sans affecter les autres, tant que leurs interfaces restent inchangées. Un des modèles les plus répandus est le modèle, baptisé OSI de l'ISO, à sept couches : physique, liaison de données, réseau, transport, session, présentation et application. Nous utilisons ce modèle pour situer les différents protocoles dédiés au streaming dans la couche correspondant à leurs fonctionnalités (cf. Figure 1.11). Nous nous focalisons sur les protocoles normalisés par l'IETF suivants : RTP, RTSP, SIP, RTCP et RSVP.

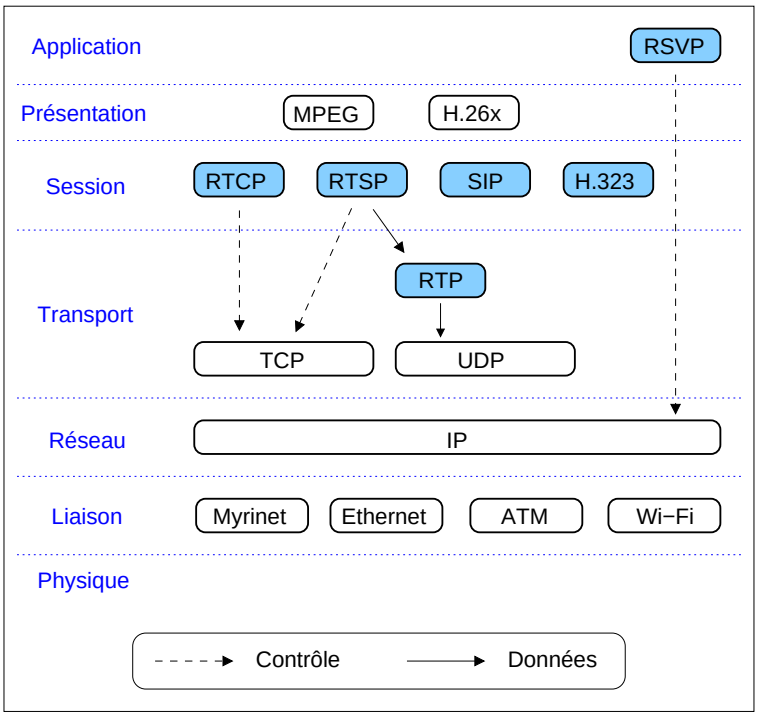


Figure 1.11 – Classement des protocoles de streaming dans les couches du modèle OSI.

Protocoles de transport

Dans les réseaux basés sur IP, deux protocoles de transfert de données sont principalement utilisés : TCP et UDP. À l'opposé du protocole UDP, qui introduit des pertes ou une arrivée désordonnée des paquets, le protocole TCP garantit la livraison des données en les expédiant à nouveau si nécessaire et effectue un contrôle de congestion lors de la transmission des données [MSM97].

Un contenu audiovisuel peut tolérer la perte d'un certain pourcentage de ses données, sans pour autant affecter la qualité perçue par le spectateur, ceci sous réserve que les données perdues ne soient pas consécutives. Ainsi, la fiabilité de livraison des données dynamiques, à l'opposé des données statiques, est souvent inutile. D'ailleurs, il ne sert à rien de retransmettre un paquet perdu si le paquet retransmis arrive lorsque la séquence à laquelle il appartient a déjà été utilisée. Non seulement la retransmission est inutile mais elle consomme des ressources, au risque de retarder d'autres paquets. Par ces faits, l'utilisation du protocole UDP convient mieux au transfert des données audiovisuelles que TCP. Cependant, UDP ne fait pas de contrôle de congestion et il n'est pas optimisé pour le transfert des données ayant des contraintes temps réel. Ces deux inconvénients sont contournés en utilisant des protocoles compatibles avec TCP [TZ01], pour le premier, et un protocole spécialisé dans le transfert des données audiovisuelles, nommé RTP [SCFJ03], pour le second.

Protocoles compatibles avec TCP Un inconvénient majeur du protocole UDP est qu'il n'effectue pas de contrôle de congestion d'une manière compatible avec TCP. En effet, il continue à augmenter le débit de transfert jusqu'à saturation du réseau. Il ne partage pas équitablement la bande passante avec les trafics bâtis sur TCP. Ceci conduit à l'effondrement du réseau en privant toutes les autres applications de bande passante (tout protocole confondu).

Dans le cas du streaming, le contrôle de la congestion revient à superviser le débit. Le contrôle du débit (« rate control ») peut être géré au niveau de l'émetteur (serveur), du récepteur (client), ou des deux (hybride). Des mécanismes de contrôle et de régularisation du débit issu du trafic UDP ont été développés pour le rendre compatible avec son homologue TCP [TZ01, FHPW00]. Un trafic est considéré compatible avec TCP, ou « TCP-friendly », s'il ne réduit pas, à long terme, le débit d'autres trafics plus que « sa version TCP » ne le réduirait dans les mêmes conditions.

Protocole de transmission temps réel RTP C'est un protocole optimisé pour le transfert de données dynamiques sur un réseau point à point tout en respectant leurs contraintes de temps réel. Il est également utilisé pour la multidiffusion. Ce protocole effectue un découpage intelligent des données pour que chaque paquet soit décodable indépendamment des autres. Les paquets RTP sont marqués temporellement de manière à être réorganisés, en cas de réception désordonnée ou tardive, afin d'afficher le stream de manière cohérente et synchronisée chez l'utilisateur. Ces marques temporelles permettent également la détection d'éventuelles pertes de paquets. RTP ne gère pas la réservation de ressources réseau et ne garantit pas de qualité du service temps réel. On y adjoint le protocole RTCP, présenté plus loin, pour effectuer le contrôle des données.

Protocoles de session

Les protocoles situés dans cette couche surveillent le transfert des données et génèrent des rapports de diagnostics dont le but est de prévenir certaines anomalies. Nous décrivons les deux protocoles principaux : RTCP [Hui03, FCC03] et RTSP [SRL98].

Protocole de contrôle temps réel, RTCP Ce protocole de contrôle est conçu pour fonctionner conjointement avec RTP. Il permet le « monitoring » de manière extensible des données transférées et fournit des fonctionnalités de contrôle et d'identification. Les paquets RTCP, envoyés périodiquement, ne comportent pas de données audiovisuelles mais contiennent des statistiques utiles à l'application telles que le taux de perte, le débit observé, la gigue, etc. Toutefois, RTCP ne spécifie pas comment l'application devrait réagir à ces rapports. Les applications sont donc libres de choisir leur manière de s'adapter vis-à-vis des rapports RTCP.

Le protocole de streaming temps réel, RTSP Ce protocole est destiné à satisfaire les critères d'efficacité d'acheminement des données continues sur des réseaux IP. Il utilise RTP pour former et transférer les paquets contenant les données multimédias. Ce protocole permet de contrôler les streams en offrant des fonctionnalités typiques d'un lecteur audiovisuel telles que lecture, pause et navigation temporelle. Il est à noter que les solutions les plus répandues actuellement, qu'elles soient de RealNetworks®, d'Apple®ou de Microsoft®, ont toutes opté pour RTSP. Notons également qu'il existe d'autres protocoles équivalents à RTSP comme SIP [RSC+02] de l'IETF ou la norme H.323 [SC01, MMS00] de l'ITU-T, qui sont dédiés aux sessions de visioconférence.

Protocoles d'application

Le protocole de réservation de ressources RSVP [ZBHJ97, TBVZ00] permet aux applications de réserver dynamiquement des ressources dans un réseau basé sur IP, afin de satisfaire leurs besoins en termes de bande passante nécessaire, délais de transfert autorisé, etc. Il émet périodiquement des messages de réservation, ou de libération, des ressources afin de maintenir dynamiquement les chemins utilisés. Ces messages, destinés aux routeurs, circulent le long du chemin qu'emprunterait le trafic des données à travers le réseau. Notons qu'il existe d'autres protocoles de réservation de ressources comme DiffServ [BBC+98], etc.

1.3.3 Serveur de streaming

Le serveur occupe la position central d'un système de streaming. Il joue un rôle essentiel en essayant d'offrir un service de streaming de qualité qui se traduit par sa faculté à garantir des streams non saccadés à toutes les requêtes admises. Pour cela, il doit tenir compte de la nature des streams, qui est très volumineuse, gourmande en bande passante et impose des contraintes strictes en matière de délai d'acheminement des données aux clients. Une machine, hébergeant un serveur de streaming doit être capable de supporter un grand nombre de clients, en tout cas, elle doit être capable de supporter le nombre désiré de clients. Elle doit donc avoir une bande passante de sortie très importante. En effet, la bande passante requise pour diffuser des données audiovisuelles en continu est, à l'instar du coût, proportionnelle à l'importance de l'audience.

D'ailleurs, une machine hébergeant un serveur de streaming doit posséder également un grand espace de stockage capable de fournir les données avec des débits très élevés.

Dans cette section, nous synthétisons les fonctionnalités principales d'un serveur de streaming à l'aide d'un modèle à cinq composants [Mos04] : une interface d'administration, un ordonnanceur, un gestionnaire de ressources, un gestionnaire de stockage et un gestionnaire de transmission (voir Figure 1.12). Au passage, nous identifions et comparons plusieurs techniques qui interviennent dans les différents composants. Il est à noter qu'il n'existe pas de technique optimale propre à chaque composant, mais plutôt un compromis global. La conception d'un serveur de streaming s'apparente donc à un exercice d'équilibre structurant ces différents modules. En fin de section, nous dressons un panorama des serveurs les plus présents dans le monde du streaming d'aujourd'hui.

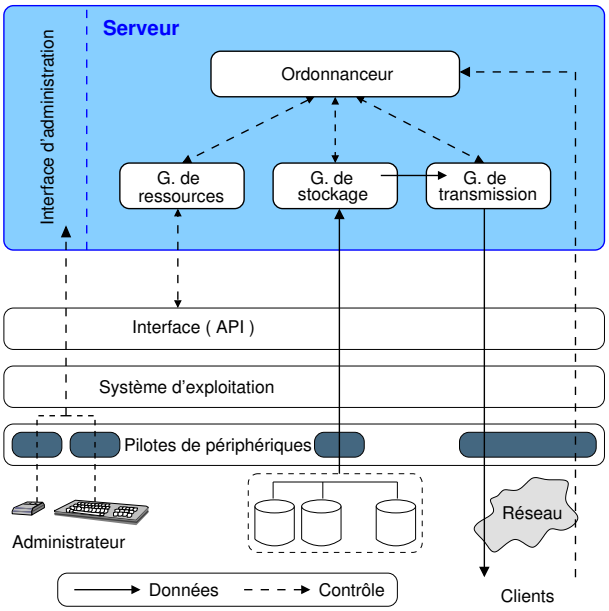


Figure 1.12 – Composants d'un serveur de streaming.

Avant de se pencher sur les détails fonctionnels du serveur et les rôles respectifs de ses composants, citons les principaux critères utilisés pour évaluer et comparer la performance des différents serveurs :

1. **La capacité du serveur** (« throughput ») : Elle est définie par le nombre de streams que le serveur est capable de gérer simultanément. Cette capacité est fonction des ressources mises à sa disposition. Selon la capacité du serveur on distingue les serveurs à petite échelle (« small-scale servers »), à échelle moyenne (« medium-scale server ») ou bien à grande échelle (« large-scale servers »). On trouve les serveurs à petite échelle dans les avions, les hôtels, etc. Les serveurs à échelle moyenne sont capables de servir jusqu'à 50 streams simultanés sur un réseau local. Quant aux serveurs à grande échelle, on parle de milliers des streams simultanés sur un réseau métropolitain, voire sur Internet. Si une technique donnée permet au serveur d'avoir une capacité plus élevée, pour les mêmes ressources, que d'autres techniques, elle est considérée comme une meilleure solution.
2. **Coût par stream** : C'est le coût total du serveur, y compris celui de la plate-forme matérielle l'hébergeant, divisé par sa capacité.

3. **Latence de démarrage d'un stream** (« startup latency ») : C'est le temps écoulé entre l'instant d'arrivée d'une requête au serveur et l'instant où le stream commence à être restitué à l'usager qui a lancé la requête (voir Figure 1.6). Lorsqu'un usager est connecté, *via* un réseau local, à un serveur à échelle moyenne, il prévoit une latence de démarrage de l'ordre de quelques secondes, alors qu'il tolère des latences plus élevées lorsqu'il est connecté à un serveur à grande échelle.

1.3.3.1 Modèle fonctionnel du serveur

Afin d'illustrer les différentes tâches qu'un serveur effectue en réponse à des requêtes demandant l'accès en streaming à des séquences audiovisuelles, nous utilisons un modèle fonctionnel à cinq composants : une interface d'administration, un ordonnanceur, un gestionnaire de stockage, un gestionnaire de ressources et un gestionnaire de transmission. Ces composants travaillent en collaboration dans le processus d'acheminement des données audiovisuelles (voir Figure 1.13). Un problème survenant dans n'importe quel composant peut donc dégrader la performance de toute la chaîne. Notons que ce modèle fonctionnel fait abstraction de tout choix d'architecture matérielle ou d'implémentation, et ne se focalise pas particulièrement sur des techniques adaptées à une classe spécifique d'application de streaming.

Un serveur de streaming doit traiter simultanément une multitude de streams sous des contraintes temporelles strictes. Pour ce faire, son fonctionnement interne est basé sur une allocation périodique des ressources aux différentes requêtes acceptées dans le serveur, de façon à assurer à chacune d'elles une continuité de service. La Figure 1.13 schématise ce fonctionnement. La période de service est appelée *cycle de service*. Durant un cycle, le serveur transmet, pour chaque stream, des données de façon à ce que la continuité des streams soit assurée. Il veille ainsi à ce que la quantité de données transmises à chaque client ne soit pas inférieure à la quantité de données nécessaires pour une restitution sans saccade du contenu audiovisuel. Les données sont récupérées depuis le système de stockage, puis placées temporairement dans la mémoire, et enfin transmises vers les clients. La taille de la mémoire ainsi que le débit maximal avec lesquels le système de stockage peut délivrer des données conditionnent, comme nous allons voir dans la Section 1.3.3.4, la longueur du cycle de service d'un serveur et déterminent par conséquent sa capacité. Notons qu'un gestionnaire de mémoire est nécessaire pour optimiser les accès à celle-ci. Les techniques d'optimisation d'accès à la mémoire, appelée également *cache*, seront examinées en détail dans la Section 1.4.

1.3.3.2 Interface d'administration

Elle représente l'interface de gestion et de configuration du serveur. Elle permet de configurer le serveur en choisissant les techniques employées pour les différents composants. Afin de suivre le comportement du serveur en temps réel, elle fournit des moyens pour placer des points de surveillance dans les différents composants. Elle donne des statistiques concernant tous les composants auxquels un point de surveillance est associé. Ces statistiques sont utilisées pour évaluer la performance des différentes techniques de gestion et d'ordonnancement.

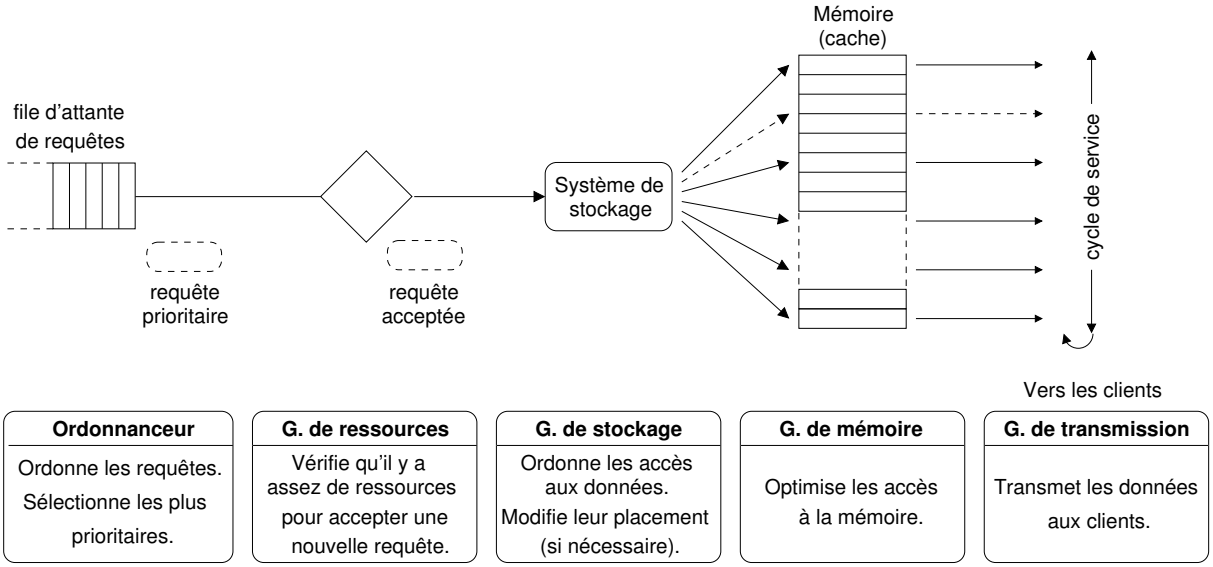


Figure 1.13 – Fonctionnalités d'un serveur de streaming.

1.3.3.3 Ordonnanceur

Si le gestionnaire de ressources juge que les ressources disponibles à un moment donné ne suffisent pas pour accepter une nouvelle requête, l'usager demandant l'accès à un contenu donné doit attendre la libération des ressources avant que sa requête soit acceptée par le serveur. Plus le temps d'attente est long, plus forte est la probabilité que l'utilisateur se rétracte de sa demande. Il incombe à l'ordonnanceur d'établir une sorte de priorité sur les différentes requêtes et de décider quelle sera la prochaine requête à servir, lorsque suffisamment de ressources se trouvent disponibles pour gérer un nouveau stream.

Par ailleurs, l'accès aux contenus audiovisuels n'est pas uniforme, c'est-à-dire qu'il est fréquent qu'une grande partie des spectateurs accède à un ensemble restreint de données, appelées *données chaudes*. En prenant en compte cet accès biaisé aux contenus audiovisuels, l'ordonnanceur cherche à augmenter la capacité du serveur. Pour cela, il regroupe, autant que faire se peut, les requêtes demandant l'accès à un contenu donné et les traite comme une seule requête diffusée ensuite en mode mutlipoint. Néanmoins, l'efficacité de ce regroupement n'est effective que si les utilisateurs tolèrent un délai, plus ou moins long, avant la restitution du contenu demandé.

Nous examinons, dans cette section, les techniques utilisées pour ordonner, regrouper et définir une priorité sur les requêtes. Nous commençons par introduire les deux paradigmes qui modélisent la rétractation des utilisateurs. Nous présentons ensuite les objectifs recherchés par les techniques d'ordonnancement de requêtes. Une taxinomie de ces techniques clôt cette section. Notons que l'ordonnanceur ne se limite pas à l'ordonnancement des requêtes ; il est également responsable de la gestion interne du serveur. Néanmoins, cette dernière est fortement liée à la fois à la plate-forme matérielle utilisée et à l'environnement de développement adopté. Ainsi, par souci de généralité, elle ne sera pas traitée.

Modèles de rétractation des spectateurs

Lorsqu'un utilisateur demande l'accès à un contenu audiovisuel, son comportement suit un des deux paradigmes suivants [DSS94] :

- **Attente minimale assurée** : tous les usagers sont supposés vouloir attendre au moins T_{min} secondes, puis leur temps d'attente est distribué exponentiellement. Ainsi, le temps d'attente d'un spectateur est $T_{att} = T_{min} + T(e^{-t})\ ,\ t > 0$.
- **Attente limitée** : dans ce paradigme, les usager n'acceptent d'attendre que si un temps d'attente maximal, T_{max}, est garanti.

Ainsi, dans une application de type N-VOD, afin d'assurer une probabilité de rétractation nulle pour les films très sollicités, appelés *films chauds*, il suffit de les diffuser tous les T_{min} secondes.

Objectifs d'ordonnancement de requêtes

Les objectifs de l'ordonnancement de requêtes sont les suivants :

- Minimiser le temps moyen d'attente des utilisateurs.
- Minimiser la probabilité de rétractation à long terme, c'est-à-dire minimiser le ratio du nombre des utilisateurs qui se rétractent divisé par le nombre total des utilisateurs.
- Minimiser la probabilité de rétractation en période de pointe à court terme : c'est-à-dire minimiser le ratio du nombre des utilisateurs qui se rétractent dans une période courte divisé par le nombre des utilisateurs qui sont connectés pendant cette période.
- Assurer l'équité entre les contenus. Intuitivement, une politique d'ordonnancement est équitable si la probabilité de rétractation pour tous les contenus est la même. Or, le nombre de requêtes demandant l'accès à un contenu « froid » peut être très faible. La probabilité de rétractation concernant ce contenu et donc la non-équité peut être difficile à estimer. Ainsi, les contenus sont groupés en N groupes de façon à ce que chaque groupe reçoive le même nombre de requêtes. La non-équité (« unfairness »), ou la variance de la probabilité de rétractation U, entre les groupes peut ensuite être mesurée. Notons P_i, la probabilité de rétractation à long terme du i^{e} groupe, P la probabilité moyenne de rétractation à long terme de tous les groupes. La non équité entre les groupes peut être exprimée ainsi :

$$U = \sum_{i=1}^{N} \frac{(P_i - P)^2}{N} \tag{1.1}$$

Ordonnancement de requêtes (« batching »)

Si les requêtes acceptées sont servies immédiatement, on aura des fluctuations dans le temps d'attente expérimenté par les utilisateurs et par conséquent des périodes avec des probabilités de rétractation élevées. Les techniques d'ordonnancement des requêtes, appelées « *batching* » dans la littérature anglophone, cherchent à pallier cet inconvénient en servant les requêtes acceptées avec un peu de retard, qui est souvent utilisé pour faire passer une séquence à vocation publicitaire. Ce retard est utilisé pour regrouper les requêtes demandant l'accès au même contenu, augmentant ainsi la capacité de serveur en servant plusieurs requêtes en un seul stream diffusé en mode multipoint.

La variable permettant le paramétrage de ce retard est la taille de la fenêtre temporelle au sein de laquelle les requêtes sont regroupées. Plus celle-ci est grande, plus le nombre de requêtes ayant des chances d'être regroupées est grand, optimisant ainsi l'utilisation des ressources dont dispose le serveur, mais augmentant le temps d'attente des utilisateurs. Toutefois, des temps d'attente élevés augmentent la probabilité de rétractation de la part des utilisateurs. Toute la subtilité du regroupement consiste donc à trouver une taille de fenêtre offrant le meilleur compromis entre l'optimisation de l'usage des ressources et la satisfaction des utilisateurs. Ce retard peut être contrôlé explicitement ou implicitement. Dans le contrôle explicite, appelé *attente forcée*, un temps d'attente minimum est fixé. Ce dernier est un paramètre critique et difficile à estimer correctement. Dans le contrôle implicite, un nombre maximum de requêtes à servir dans un intervalle fixe de temps est déterminé au préalable. Le contrôle implicite apporte une meilleure performance dans la mesure où il induit une probabilité de rétractation plus faible à long terme même si cette probabilité reste élevée dans une période de pointe [WCM02].

Les requêtes sont ainsi amenées à attendre un laps de temps afin d'être regroupées. L'ordonnanceur établi ensuite l'ordre selon lequel les groupes seront servis dès que suffisamment de ressources se trouveront disponibles pour prendre en charge un nouveau stream. De nombreuses politiques d'ordonnancement ont ainsi été proposées pour définir le groupe qui sera servi en priorité [AWY01, Sar04, SQ07]. Ces politiques restent des variantes de trois politiques principales que nous présentons ci-dessous :

Politique FCFS Dans la politique FCFS, nommée « premier arrivé premier servi », toutes les requêtes sont mises dans une file d'attente, selon leur ordre d'arrivée. Dès la libération de ressources suffisantes, l'ordonnanceur choisit immédiatement la requête en tête de file, celle qui a attendu le plus, pour être servie. Toutes les requêtes de la file, accédant au même contenu que la requête de tête, sont sélectionnées pour le regroupement. Cette politique met tous les contenus sur un plan d'égalité car aucune priorité n'est assignée aux requêtes accédant aux contenus chauds.

Politique MQL La politique MQL (« Maximum Queue Length ») affecte à chaque contenu disponible dans le serveur une file d'attente dans laquelle les requêtes y demandant l'accès sont mises en attente. La sélection des requêtes à servir, après libération de ressources, est basée sur la cardinalité des files d'attente. La file contenant le plus de requêtes en attente est servie en priorité. Cette politique prend en considération la fréquence d'accès aux différents contenus et favorise le service des contenus chauds. Néanmoins, dans une configuration de serveur à capacité limitée, les contenus froids, correspondants aux files d'attente de faible cardinalité, courent le risque de ne pas être servis ou d'être servis après des temps d'attente importants. Ceci accroît la probabilité de rétractation de la part des utilisateurs qui cherchent à accéder à ces contenus et accentue par conséquent la non équité entre les contenus.

Politique FCFS-n Comme nous l'avons mentionné, la politique FCFS ne favorise pas explicitement les accès aux contenus chauds. Afin de pallier cet inconvénient, la technique nommée FCFS-n réserve à l'avance les ressources nécessaires pour servir périodiquement les n contenus les plus demandés. Ainsi, les contenus les plus chauds sont examinés périodiquement, toutes les B minutes par exemple. Durant chaque période B, appelée période de regroupement, les requêtes demandant l'accès aux contenus chauds sont regroupées. Les requêtes demandant l'accès

aux contenus froids sont, quant à elles, servies en FCFS classique. Notons qu'avec une valeur de n égale à 0, la politique FCFS-0 est réduite à une politique FCFS simple. Ainsi, les ressources du serveur sont divisées en deux catégories : les ressources réservées pour les contenus chauds (n contenus) et les ressources allouées à la demande pour les contenus froids. L'efficacité de cette politique réside alors dans le partitionnement judicieux des ressources à travers un bon choix du paramètre n.

Ainsi, FCFS et FCFS-n peuvent garantir un temps d'attente maximal pour les différentes requêtes, tandis que MQL ne le peut pas. Cela est dû au fait que MQL ne prend pas en compte la durée d'attente d'un utilisateur. Enfin, même si dans les serveurs à capacité limitée MQL présente une faible probabilité de rétractation à long terme par rapport à FCFS, leurs performances s'inversent dans les serveurs à capacité élevée [DSS94].

1.3.3.4 Gestionnaire de ressources

Étant donné que les ressources mises à disposition du serveur (espace mémoire, bande passante du système de stockage, etc.) sont limitées, ce dernier ne peut servir simultanément qu'un nombre limité de requêtes. Le gestionnaire de ressources est le composant chargé d'exercer un *contrôle d'admission* vis-à-vis des nouvelles requêtes lors de leur soumission. Ce contrôle sert à vérifier que le nouveau stream produit par l'acceptation d'une nouvelle requête n'altérera pas les stream déjà admis. Il s'assure ainsi que le serveur possède suffisamment de ressources (processeur, mémoire, débit de transfert du système de stockage, bande passante réseau, etc.) pour supporter simultanément l'ensemble des streams tout en respectant les contraintes imposées par ces streams. Ces contrôles affectent tous les dispositifs de la plate-forme hébergeant le serveur et se répercutent jusqu'au système de stockage. Par souci de généralité, nous ne présentons que l'impact du contrôle d'admission sur la taille de la mémoire disponible dans le serveur et sur le débit de transfert du système de stockage.

Supposons qu'à un instant t donné, le serveur supporte n streams, c'est-à-dire qu'il assure un débit constant égal à $r_0, \cdots, r_{n-1}$ bits par seconde pour chacun des streams $S_0, \cdots, S_{n-1}$. Soit S_n, le stream demandé par une nouvelle requête r_n soumise au serveur au même instant t, soit R_n son débit, soit T la longueur du cycle de service. Pendant un cycle T, chaque stream S_i consomme $(T \times r_i)$ bits. Ceci implique qu'avant le début de chaque cycle, $(T \times r_i)$ bits doivent être chargés depuis le système de stockage vers la mémoire.

Supposons que les blocs de données sont de taille b et notons M_{total} la taille de la mémoire totale mise à la disposition du serveur pour le traitement des streams, exprimée en nombre de blocs. L'ensemble de la mémoire consommée par tous les streams, pendant un cycle T, ne doit pas excéder la mémoire mise à disposition du serveur. On peut donc énoncer la première condition, relative à la taille de la mémoire, pour l'acceptation de la requête R_n comme suit :

$$\sum_{i=0}^{n} \left(\left\lceil \frac{T \times r_i}{b} \right\rceil + 1 \right) \leq M_{total} \tag{1.2}$$

Si le débit des streams est une caractéristique fixe de ces derniers, donc indépendant de la conception du serveur, la durée du cycle est, en revanche, fixée de manière à assurer un bon équilibre entre le nombre total des streams supportés simultanément et la taille mémoire totale disponible dans le serveur. Plus la période T est longue, plus la taille de la mémoire nécessaire pour chaque stream est importante.

Notons T_{achem} le temps d'acheminement d'un bloc de données du système de stockage vers la mémoire, comprenant le temps de lecture. Le temps total nécessaire à l'acheminement de tous les blocs de données des requêtes acceptées ne doit pas dépasser la durée du cycle de service. La deuxième condition, relative au débit de transfert du système de stockage, pour l'acceptation de la requête R_n s'écrit donc :

$$\sum_{i=0}^{n} \left(\left(\left\lceil \frac{T \times r_i}{b} \right\rceil + 1 \right) \times T_{achem} \right) \leq T \tag{1.3}$$

Comme pour la condition (1.2), T_{achem} est une caractéristique fixe du système de stockage sous-jacent. Il s'agit ici de déterminer de façon optimale la durée du cycle T de manière à pouvoir gérer un maximum de streams simultanés tout en prenant en compte les contraintes imposées par la mémoire mise à disposition du serveur (condition 1.2), d'une part, et par le débit de transfert du système de stockage (condition 1.3), d'autre part.

1.3.3.5 Gestionnaire de stockage

Le système de stockage est sans doute une partie fondamentale de la machine hébergeant un serveur de streaming puisqu'il est à la base du stockage et de l'envoi des données audiovisuelles. Les tâches dévolues au gestionnaire de stockage sont : le placement et la répartition des données au sein du système de stockage ainsi que l'ordonnancement d'accès à ces données [HGG+04]. Dans les paragraphes suivants, nous survolons les différentes techniques employées par ce composant afin de maximiser l'efficacité du système de stockage, le rendre tolérant aux pannes et s'assurer que les streams seront servis à un rythme garantissant leur continuité. Nous nous focalisons sur les techniques conçues pour optimiser l'utilisation d'un système de stockage composé d'un ou de plusieurs disques magnétiques. Ceci est du au fait que ces derniers sont les meilleurs candidats pour le dispositif de stockage au sein d'un serveur à la fois performant et rentable. En effet, ils répondent le mieux aux contraintes du serveur de streaming en termes de capacité de stockage, de débit de transfert, de temps d'accès ainsi que de coût de dispositif.

Placement de données

Le temps d'acheminement de blocs de données dépend de leur localisation dans les différentes zones d'un disque. En effet, un disque magnétique est composé d'un ensemble de plateaux. Sur chaque plateau on trouve un certain nombre de pistes concentriques (voir Figure 1.14). Les pistes intérieures du disque contiennent moins de données que les pistes extérieures. Puisque le disque tourne à une vitesse angulaire constante, le débit de lecture de données diffère entre les pistes [Zim98]. En effet, si la tête de lecture se trouve sur une piste intérieure, elle lira moins de données que si elle se trouve sur une piste extérieure. Afin de réduire cette différence de débit, les fabriquant des disques regroupent plusieurs pistes, ayant la même capacité de stockage, dans une zone logique, on parle alors de « multi-zoning ». Le débit de transfert de chacune des zones est ensuite réduit à celui de la piste la plus lente contenue dans la zone. Une politique d'optimisation de placement des données dans un disque devrait prendre en compte cette variabilité du débit entre les différentes zones.

L'objectif recherché par une politique de placement des données est de réduire le temps de cheminement des blocs de données, T_{achem}, de façon à garantir le respect de leur continuité.

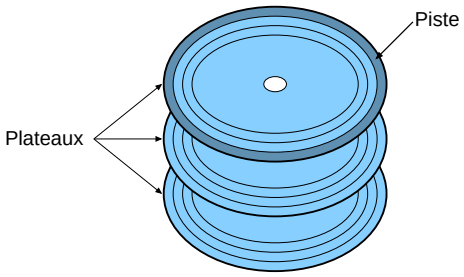


Figure 1.14 – Géométrie d'un disque dur magnétique.

Parmi les politiques proposées dans la littérature pour le placement des blocs de données sur un seul disque, on peut citer : la politique EVEREST [GIZ96, GZS+97], la politique organ-pipe [AS95] et la politique REBECA [AOG92].

La politique EVEREST place les blocs de données d'un même fichier de manière contiguë sur le disque minimisant ainsi le temps d'acheminement de ces blocs en cas de lecture séquentielle. La politique organ-pipe, contrairement à Everest, minimise le temps d'acheminement des blocs de données entre des accès successifs à plusieurs fichiers. Elle place, périodiquement, les blocs les plus fréquemment demandés à la périphérie du disque. La politique REBECA (« region-based block allocation ») minimise le temps d'acheminement en minimisant les délais d'accès. Elle divise d'abord le disque en un nombre fixe de régions contiguës concentriques. Les blocs successifs du même fichier sont ensuite placés dans des régions adjacentes.

Répartition de données

Afin de s'affranchir de la limitation du débit de transfert d'un seul disque et de permettre donc au serveur de supporter un maximum de requêtes simultanées, le système de stockage est classiquement composé de plusieurs disques appelés *matrice à disques* (« disk array »). La capacité de stockage et le débit de transfert de la matrice augmentent proportionnellement avec le nombre de disques qu'elle contient. En revanche, sa durée moyenne de fonctionnement avant défaillance (« Mean Time To Failure ») est beaucoup plus faible que celle d'un seul disque. Ainsi, l'un des objectifs du gestionnaire de stockage est de répartir les données sur les différents disques de manière à rendre le système de stockage tolérant aux pannes des disques. Une politique de répartition des données cherche également à équilibrer l'utilisation des différents disques de façon à éviter qu'un disque soit un goulot d'étranglement. Pour cela, elle doit, tout comme une politique de placement des données, prendre en compte la popularité de celles-ci, c'est-à-dire la probabilité d'y accéder. On distingue deux approches de répartition des données dans la littérature [GB00] : les données sont dupliquées sur plusieurs disques ou bien elles sont distribuées sur plusieurs disques.

Duplication de données La première approche, analogue au RAID1, est basée sur l'utilisation de disques miroirs [BG88]. Elle garantit un recouvrement instantané en cas de panne d'un disque et augmente le débit de transfert global du système de stockage. En contrepartie, l'espace requis pour la réplication est très important, compte tenu du volume que représentent les données audiovisuelles. Pour pallier cet inconvénient, on peut ne dupliquer que les données

les plus fréquemment accédées. À titre d'exemple, la politique DPSR [DKS95] effectue une réplication dynamique (« copyback ») des blocs demandés fréquemment de manière à équilibrer la charge imposée par ceux-ci.

Distribution de données (Data Stripping) La deuxième approche repose sur la distribution des blocs de données sur les différents disques permettant ainsi d'équilibrer la charge sur l'ensemble des disques, de manière indépendante de la popularité des données. Des données de parité peuvent être ajoutées pour la reconstruction des données en cas de panne [PGK88]. Cette approche, analogue à RAID0, RAID4, RAID5 et RAID6, requiert un espace de stockage moindre mais nécessite un temps de calcul pour la reconstruction des données perdues.

La distribution de données peut être totale sur tous les disques ou partielle sur un sous-ensemble de disques. La distribution totale soulève l'inconvénient que la panne éventuelle de n'importe quel disque de la matrice rend indisponible (jusqu'à la reconstruction des données perdues) la totalité des données emmagasinée sur la matrice. La distribution partielle dans des sous-groupes permet de pallier cet inconvénient. En effet, les données perdues sont uniquement celles emmagasinées dans le groupe contenant le disque en panne. Elle peut également aboutir à une meilleure performance du système de stockage en regroupant les disques dont les caractéristiques (capacité de stockage, débit de transfert, etc.) sont les mêmes. D'ailleurs, elle facilite la reconfiguration en cas d'addition de disques au système de stockage.

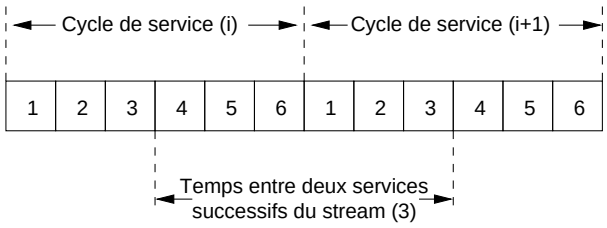
Ordonnancement d'accès aux données

Pendant un cycle de service, le gestionnaire de stockage doit alimenter la mémoire avec les données nécessaires pour que toute requête, approuvée par le gestionnaire de ressources, soit servie en quantité de données suffisante pour une restitution sans saccade. Afin d'augmenter l'efficacité du système de stockage, il est primordial d'ordonner, de manière aussi optimale que possible, les accès aux données au sein de celui-ci. Dans la suite, nous examinons uniquement les algorithmes d'ordonnancement d'accès aux données dans le cas où celles-ci sont placées sur un seul disque. Néanmoins, notons que selon le modèle de répartition utilisé, les techniques d'ordonnancement d'accès aux données réparties sur plusieurs disques, exploitent la possibilité d'extraire en même temps des données différentes, emmagasinées sur des disques différents.

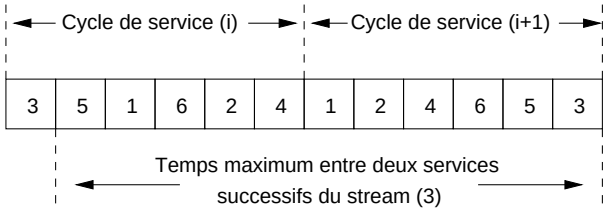
L'objectif d'une technique d'ordonnancement d'accès aux données, placées sur un seul disque, est de minimiser le temps maximum entre deux services consécutifs d'un stream donné. En effet, plus le temps entre deux services consécutifs d'un stream est grand, plus important sera le volume de données qu'il faut lire à chaque cycle, et par conséquent plus grande sera la taille de mémoire nécessaire pour assurer une restitution continue des différents streams. Traditionnellement, l'ordonnancement des accès aux données placées sur un seul disque se base sur une des trois techniques suivantes [GVK+95] : RR, SCAN et/ou EDF.

La technique RR sert les requêtes de lecture périodiquement selon un ordre défini. Cet ordre est conservé durant les différents cycles. RR permet donc de réduire le temps entre deux services consécutifs d'un stream donné à un seul cycle de service. Cependant, elle ne peut pas prendre en compte le placement physique des données. Ceci implique un temps d'accès élevé et réduit donc le débit de transfert du disque.

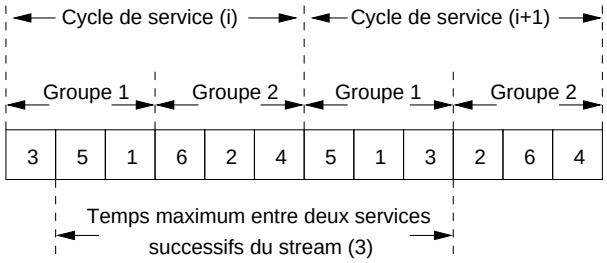
La technique SCAN ne conserve pas, durant des cycles différents, l'ordre suivant lequel les différents stream sont servis. Elle ordonne les streams selon la localisation des données sur les



(a) L'algorithme d'ordonnancement RR



(b) L'algorithme d'ordonnancement SCAN



(c) L'algorithme d'ordonnancement GSS

Figure 1.15 – Algorithmes d'ordonnancement d'accès aux données placées sur un disque.

disques, minimisant ainsi le temps d'accès. La technique SCAN, à l'opposé de RR, optimise donc l'utilisation du disque au prix d'une demande élevée de mémoire. En effet, elle impose une durée relativement importante entre deux services consécutifs d'un stream donné, allant jusqu'à deux cycles ou presque (voir Figure 1.15). Afin d'assurer la continuité du service, elle impose la réservation de deux fois plus de mémoire par rapport à RR. Les conditions d'admission (1.2) et (1.3) doivent être changées en conséquence.

Finalement, dans la technique EDF, plus orientée temps réel, les requêtes sont servies selon des priorités qui leur sont associées. La priorité d'un stream d'être servi est basée sur le délai maximal après lequel sa restitution subira des saccades. Ce délai dépend de la quantité des données disponibles dans la mémoire. Par conséquent, EDF assure un service continu des streams mais l'efficacité du système de stockage se trouve réduite car le temps d'accès aux données n'est pas pris en compte.

De très nombreuses approches combinant les techniques précédentes ont été proposées telles que SCAN-EDF [RW94] ou GSS [YCK93]. Dans SCAN-EDF, les accès sont basés, en premier, sur la priorité EDF garantissant donc les temps critiques des lectures. S'il arrive que plusieurs accès ont la même priorité, ils sont servis en SCAN, réduisant ainsi les temps d'accès. En regroupant les streams, la technique GSS combine SCAN et RR, et tire profit des avantages de SCAN en matière de bonne utilisation du disque et de RR en matière de faibles besoins

en mémoire. Entre les groupes, RR est appliquée, fixant par conséquent l'ordre selon lequel les différents groupes sont servis. Au sein de chaque groupe, SCAN est utilisé, réduisant ainsi les temps d'accès à l'intérieur de chaque groupe (voir Figure 1.15). GSS peut donc être considérée comme un compromis entre l'optimisation du temps d'accès et la réduction de la taille de la mémoire nécessaire. Si un seul groupe est utilisé, le GSS se réduit à un ordonnancement SCAN. Si un seul stream est assigné à un groupe, GSS se réduit à un ordonnancement RR.

1.3.3.6 Gestionnaire de transmission

Ce composant est responsable du streaming proprement dit : il gère la transmission des blocs de données aux clients à travers le réseau. Le gestionnaire de transmission fait donc usage de protocoles dédiés au transfert de données possédant des contraintes de temps réel, comme ceux présentés dans la Section 1.3.2.2. Rappelons que la retransmission d'un paquet est effectuée uniquement si le paquet retransmis peut arriver avant qu'il soit temps de le décoder et de le restituer. Sinon, la retransmission est non seulement inutile mais risque de retarder l'arrivée d'autres paquets. Ainsi, le gestionnaire de transmission est chargé d'optimiser la transmission en employant des techniques d'adaptation de contenus à l'état du réseau. Enfin, le gestionnaire de transmission est amené à se servir de techniques permettant la protection des données contre les pertes et/ou les transformations possibles lors d'une transmission.

Streaming adaptatif

Dans le but de répondre au problème de variabilité de la bande passante des réseaux, le gestionnaire de transmission adapte son taux de transmission au débit de transmission dicté par les conditions du réseau. Deux types d'adaptations ont été proposés dans la littérature, l'un est dit *statique* et l'autre est dit *dynamique*.

Adaptation statique Plusieurs versions alternatives d'un même contenu, encodées à des débits différents, sont mises à disposition des utilisateurs pour s'adapter aux conditions de réseaux. Le débit de chaque version est correctement dimensionné pour être transmis sur une connexion donnée. La sélection entre les versions est effectuée par un utilisateur, avant le démarrage du streaming, en fonction de la connaissance qu'il a de la capacité de sa connexion. Bien que cette méthode soit facile à mettre en œuvre, elle est très gourmande en matière d'espace de stockage. Dans un serveur employant cette approche, on peut trouver des versions associées aux niveaux de qualité suivants :

- **Très basse qualité** : elle correspond à un flux audiovisuel dont le débit est inférieur à 48 Kb/s. Elle est destinée à une connexion par modem classique 56 Kb/s ou à la téléphonie mobile de 2^{e} génération (GPRS : 20 à 40 Ko/s).
- **Basse qualité** : elle correspond à un flux audiovisuel dont le débit est inférieur à 160 Kb/s. Elle est destinée aux connexions ADSL de bas débit.
- **Qualité moyenne** : elle correspond à un flux audiovisuel dont le débit est aux alentours de 320 Kb/s. Elle est destinée aux connexions ADSL normales (150 à 200 Ko/s) et aux mobiles de 3^{e} génération (UMTS : 150 à 200 Ko/s). Pour ces derniers la qualité est très bonne, car la taille de l'écran est plus petite que celle d'un ordinateur conventionnel.

- **Bonne qualité** : elle correspond à un flux audiovisuel dont le débit est environ 500 Kb/s. Elle est destinée aux connexions ADSL haut débit (4 Mb/s).

- **Haute qualité** : elle correspond à un flux audiovisuel dont le débit est supérieur à 1 Mb/s. Elle est destinée aux connexions ADSL très haut débit (supérieur à 8 Mb/s).

Adaptation dynamique Le gestionnaire de transmission se charge de diffuser des streams adaptés en temps réel aux différentes connexions des utilisateurs. Au fur et à mesure de la diffusion, des données de contrôle sont envoyées au serveur pour l'informer des conditions de réception du stream. Le gestionnaire de transmission utilise ces données pour estimer la bande passante disponible et s'adapte en conséquence. Si la connexion se détériore, causant une baisse du taux de transfert, le contenu continuera à être livré mais avec une moindre qualité évitant ainsi des interruptions de diffusion. Si en revanche la connexion devient plus fluide, la qualité du contenu s'améliorera.

L'adaptation peut être spatiale, en réduisant le nombre de couleurs et/ou les dimensions des images dans une vidéo par exemple, ou temporelle, en réduisant par exemple le nombre des images affichées par seconde. La conception de tels mécanismes d'adaptation est compliquée par des exigences contradictoires : il faut réagir rapidement aux changements fréquents du réseau, mais on souhaite des changements peu fréquents dans la qualité des streams perçue par le spectateur. On distingue quatre méthodes pour adapter dynamiquement les streams :

- **Commutation entre versions**

  Dans cette approche appelée aussi « Multiple File Switching », le serveur contient plusieurs versions, associées à différentes qualités pour chaque contenu audiovisuel, comme dans le cas de l'adaptation statique. Des techniques telles que *Intelligent Streaming* de Microsoft® ou *SureStream* de RealNetworks®, permettent de commuter en temps réel entre les versions. Cette dernière recommande le calcul suivant pour choisir la version à streamer :
  - 75% de la bande totale disponible pour les connexions analogiques telles que les modems ;
  - 90% de la bande totale disponible pour les connexions haut débit telles que DSL.

- **Transcodage du flux**

  Dans cette approche, appelée aussi façonnage du flux (« rate shaping ») ou mise en forme du flux, on vise à ajuster le débit du stream en l'encodant à nouveau [WAF99]. L'inconvénient de cette technique est qu'elle induit des calculs coûteux, mais elle nécessite moins d'espace de stockage que l'approche précédente.

- **Utilisation de codages adaptatifs**

  L'utilisation de l'encodage hiérarchique de contenu audiovisuel (cf. Section 1.1.2.2) permet d'adapter le débit d'un contenu aux conditions du réseau [MJV96]. Afin de réduire la taille du stream, et par conséquent son débit, et de s'adapter donc aux dégradations de la bande passante, on peut abandonner des couches si nécessaire, mais pas au hasard. Il faut abandonner d'abord la dernière couche d'amélioration, mais jamais la couche de base. En effet, si elle n'est pas reçue, les couches suivantes ne peuvent rien améliorer. En général, on cherche à faire passer un maximum de couches dans la bande passante disponible.

L'encodage par descriptions multiples (cf. Section 1.1.2.2) peut, lui aussi, être utilisé pour adapter dynamiquement le contenu. En effet, les descriptions peuvent être abandonnées partout où cela peut s'avérer nécessaire : dans l'émetteur si la largeur de bande est inférieure aux prévisions, dans le récepteur s'il est pas nécessaire, ou pas possible, d'utiliser toutes les descriptions reçues.

- **Transmodage du flux**
  Le transmodage est une technique réservée pour les cas extrêmes où le réseau ne permet pas de transférer les données sous leur forme initiale. Elle consiste à transformer complètement la nature des données [Per04]. Des exemples d'une telle transformation peuvent être : un discours audio en texte, une vidéo en diaporama d'images ou même en un résumé textuel, etc.

Contrôle d'erreurs de transmission

Toute donnée, audiovisuelle ou non, peut subir des pertes ou des transformations, au niveau de la couche physique, lors d'une transmission à distance. Ceci peut être provoqué par des atténuations dues au canal de transmission ou par des réflexions multiples sur des obstacles. Plusieurs techniques ont été développées pour protéger les données des erreurs, surgissant lors d'un transfert, telles que l'utilisation d'un code correcteur d'erreurs et/ou d'un encodage résistant aux erreurs.

Code correcteur d'erreurs L'utilisation d'un code correcteur d'erreurs, FEC, consiste à ajouter aux données originales des données de redondance selon des règles connues du client récepteur. Les données redondantes sont ensuite utilisées par le récepteur pour reconstituer les données originales, qu'elles soient perdues ou transformées. Ces techniques [PW99, WCK03] permettent donc d'extraire les données d'origine, même si les données se trouvent fortement altérées lors de la transmission. À titre d'exemple, nous citons le code correcteur, spécifique aux données sonores, proposé par Perkines et standardisé par l'IETF [PKH+97].

Notons que la quantité de données redondantes ajoutée est tributaire du taux de perte. Or, dans la réalité, ce taux est non seulement inconnu mais varie beaucoup avec le temps. Ce facteur, combiné avec le fait que le rendement de cette technique est du type « tout ou rien » rend cette technique très difficile à utiliser. En effet, le nombre d'erreurs est supérieur ou inférieur à celui qui était prévu. Si les pertes sont trop nombreuses, la redondance ajoutée ne sera pas suffisante et les pertes ne seront pas récupérées.

Utilisation de l'encodage par descriptions multiples Les données audiovisuelles compressées sont très sensibles aux pertes et aux erreurs de transmission. De ce fait, des encodages résistant aux erreurs (« error-resilient video coding ») de ces données ont été sollicités depuis longtemps. L'encodage par descriptions multiples (cf. Section 1.1.2.2) peut rendre les données audiovisuelles résistantes aux pertes et aux erreurs de transmission [AW01, Goy01]. En effet, la perte d'une description d'un contenu n'empêche pas sa restitution. Néanmoins, plus on reçoit de descriptions, plus grande est la qualité du contenu restitué. Notons qu'avec ce type d'encodage, il n'existe pas d'effet « tout ou rien » mais plutôt une dégradation de la qualité.

1.3.3.7 Serveurs existants

Après avoir présenté les différent aspects d'un serveur de streaming, nous clôturons cette section par une liste, non exhaustive, des serveurs de streaming les plus populaires et qui sont les principaux acteurs du monde du streaming.

SHOUTcast et Icecast : ce sont les serveurs de radio en ligne les plus populaires de l'Internet. Fondés sur la technologie de diffusion de MP3, ils ont l'avantage d'être gratuits et permettent de diffuser du son vers la plupart des lecteurs audiovisuels existants actuellement.

Helix™ : il est l'héritier d'une longue lignée de serveurs de streaming développés par RealNetworks® depuis une dizaine d'années. Fondés sur une technologie propriétaire, les serveurs Helix™ sont disponibles en version d'essai gratuite, mais avec des capacités limitées. Ils permettent de diffuser n'importe quel type de fichier audio ou vidéo. Ce sont les seuls à être compatibles avec les formats Real®.

Windows® Media Service : comme son nom l'indique, c'est un serveur développé par Microsoft®. Les dernières versions de ce logiciel sont payantes et ne fonctionnent que sur un serveur disposant de Windows® 2003 Server.

QuickTime®/Darwin Streaming Server : ce sont des serveurs de streaming à la fois audio et vidéo, fondés sur une architecture QuickTime®. Si le serveur QuickTime® est payant, son homologue Darwin a l'avantage d'être un logiciel libre et disponible gratuitement pour une multitude de systèmes d'exploitation. Ils ont aussi l'avantage d'être très simples d'emploi.

VideoLAN : c'est une solution de streaming gratuite, développée par les étudiants de l'Ecole Centrale de Paris. Ce logiciel permet de mettre en place très rapidement un flux de streaming entre deux ordinateurs distants et a l'avantage de diffuser des formats tels que le DivX®, MPEG-4, ou encore le contenu des DVD vidéo.

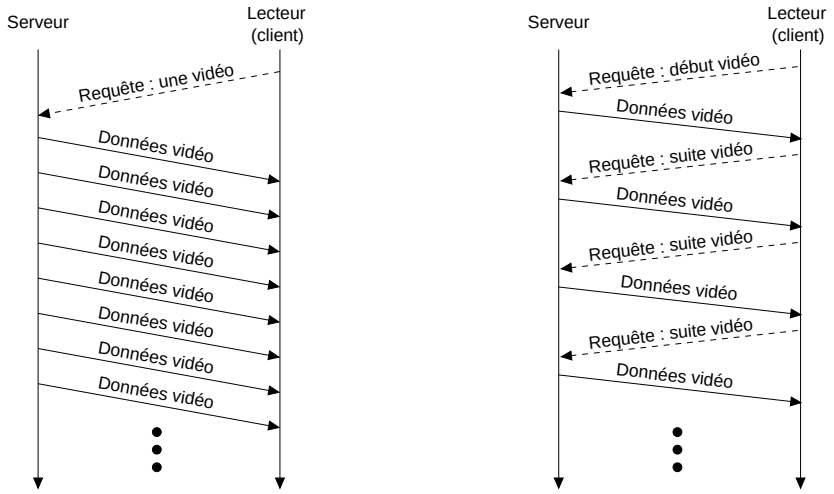
1.3.4 « Client pull » versus « serveur push »

Selon le paradigme d'interaction adopté entre le client et le serveur, on peut distinguer deux catégories de systèmes de streaming : les systèmes dits *ouverts* (« open-loop ») et les systèmes dits *fermés* (« closed-loop »). Les systèmes ouverts emploient le modèle d'interaction *offre par le serveur* (« serveur push ») tandis que les systèmes fermés emploient le modèle *demande par le client* (« client pull »).

1.3.4.1 Offre par le serveur (serveur push)

Dans cette approche, le serveur se charge de la gestion de la continuité des streams. Un client envoie une seule requête d'accès à un contenu donné et c'est au serveur de gérer et d'envoyer le stream correspondant en assurant le respect de ses contraintes de temps réel (voir Figure 1.16). La gestion de la continuité au niveau du serveur, même si elle reste une tâche complexe, permet d'exploiter la périodicité des données dynamiques et d'optimiser par conséquent l'utilisation des ressources mises à sa disposition. De ce fait, ce paradigme a été largement adopté dans les systèmes de streaming, notamment pour les applications de type diffusion en direct.

En revanche, le support des interactions utilisateur, et les fonctions de magnétoscope en particulier, sont très difficiles à mettre en œuvre dans un système employant ce modèle.



(a) Server push (b) Client pull

Figure 1.16 – « Client pull » versus « serveur push ».

1.3.4.2 Demande par le client (client pull)

À l'inverse de l'approche précédente, la gestion de la continuité des streams est laissée à la charge du client. Ainsi, l'accès aux données se traduit par un ensemble de sous-requêtes que le client envoie au serveur à chaque fois qu'il en a besoin (voir Figure 1.16). Le serveur est donc déchargé de la contrainte de temps réel qui se voit réduite à assurer des temps de réponse minimaux. En ce sens, ce paradigme ne diffère pas fondamentalement d'un serveur de fichiers classique.

Ce modèle, du fait du dialogue permanent entre le client et le serveur, génère un trafic supplémentaire. Ce trafic reste négligeable compte tenu du volume colossal des données audiovisuelles. En effet, il n'induit pas de délais supplémentaires lors de la transmission des données. Néanmoins, la capacité d'un serveur se trouve réduite de 5 à 20 % dans un système fermé par rapport à un système ouvert [RVT96]. Ceci s'explique par le fait que les opportunités d'optimisation au niveau du serveur se trouvent réduites.

En revanche, ce modèle présente plusieurs avantages. Il permet une meilleure gestion des ressources mises à disposition du client puisque ce dernier ne demande que ce dont il a besoin. Aussi, la gestion d'interactions utilisateur se trouve grandement simplifiée dans un système fermé. Ceci est particulièrement vrai pour les fonctions de magnétoscope qui se traduisent souvent par des accès à des blocs de données non consécutifs dans le stream. Enfin, il est mieux adapté aux systèmes de streaming comprenant plusieurs serveurs, distribués, car il ne nécessite pas une synchronisation entre les différents émetteurs [LW00], cette synchronisation étant effectuée par le client.

1.4 Les caches dans les systèmes de streaming

Le terme *cache* est un mot anglais signifiant un endroit où des choses peuvent être cachées ou gardées. Ce terme, utilisé en informatique pour se référer à une préservation des données pendant des périodes plus ou mois éphémères, fait référence à plusieurs sortes de caches selon leurs localisations dans un système de streaming. Un tel système est composé, comme nous l'avons vu, de trois parties : le client, le serveur et le réseau les connectant. On distingue donc trois sortes de cache, chacune ayant des rôles et des objectifs bien distincts : cache situé au niveau du client appelé *cache de lecture* (« player buffer »), cache situé au niveau du serveur appelé *cache de serveur* et enfin cache situé au niveau du réseau appelé *cache de réseau* ou *proxy* (voir Figure 1.17).

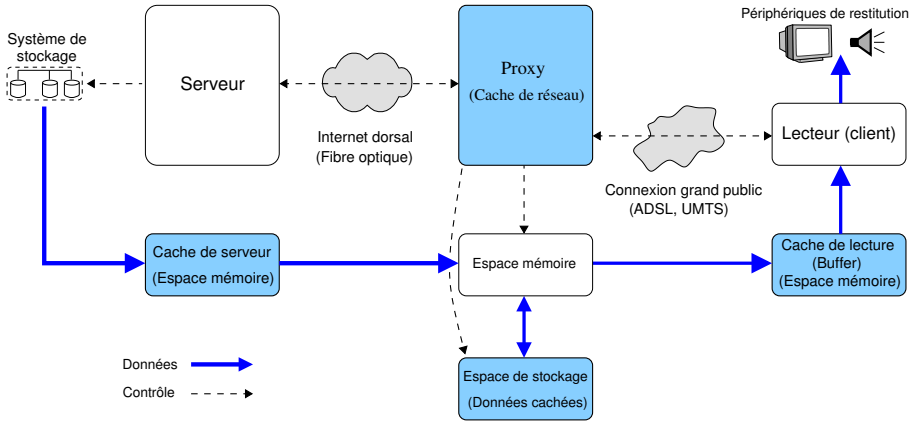


Figure 1.17 – Les caches dans un système de streaming.

Un cache, quel que soit son type, est typiquement piloté par un gestionnaire qui optimise son utilisation. Afin de mesurer l'efficacité d'une technique donnée de gestion de cache, on mesure le *taux de succès*, appelé aussi *hit rate*. Le taux de succès correspond au ratio entre le nombre de blocs [5] de données trouvés dans le cache lorsqu'ils sont demandés et le nombre total de blocs auxquels on a accédé. Une métrique alternative est le *taux d'échec*, appelé aussi *miss ratio*, qui correspond au ratio entre le nombre de blocs de données non trouvés dans le cache lorsqu'ils sont demandés et le nombre total de blocs accédés (taux d'échec = 1 − taux de succès). Plus le taux de succès est élevé lors de l'utilisation d'une technique de gestion donnée, meilleure est la technique. La gestion de caches a fait l'objet d'une abondante littérature et plusieurs politiques de gestion de caches ont été créées. Dans cette section, nous décrivons les rôles et les objectifs des différentes sortes de cache et nous détaillons quelques unes des techniques de gestion les plus représentatives.

1.4.1 Cache de lecture

Un cache, appelé « buffer », est introduit au niveau d'un client afin de restituer les streams sans saccade et de combattre les effets de la gigue présente dans les réseaux à commutation de paquets (cf. Section 1.3.2.1). Il permet donc d'apporter une meilleure qualité de service, chose à laquelle l'utilisateur est particulièrement sensible. Un cache de lecture est constitué

5. Les différents blocs de données ont la même taille, exprimée en octets.

d'une mémoire tampon (« buffer ») placée dans la machine hébergeant le client. Il sert à stocker temporairement les données audiovisuelles avant leur décodage et leur restitution à un usager. La mise en cache de données peut se décomposer en la mise en cache initiale (« prebuffering » ou « preroll ») et mise en cache durant le streaming du contenu (« buffering » ou « rebuffering »).

1.4.1.1 Mise en cache initiale

Elle sert à remplir le cache de lecture du client avant le démarrage de la restitution du contenu. Ceci permet d'absorber les saccades éventuelles issues de la perte ou de l'arrivée tardive de paquets lors de la transmission, en les récupérant à nouveau avant qu'il soit temps de les restituer. La mise en cache initiale introduit donc un délai avant le démarrage d'un stream. Or, ce délai reste négligeable et même toléré par les spectateurs. Les lecteurs actuels utilisent couramment des caches de lecture capables d'emmagasiner 5 à 15 secondes de données audiovisuelles compressées.

Afin de réduire au maximum le temps de remplissage initial du cache, certains systèmes actuels effectuent une mise en cache accélérée (« over-buffering ») en transférant les données à des taux supérieurs aux taux nominaux dictés par les streams, si le serveur et le réseau le permettent. Cette technologie est appelée « Instant-on/Always-on » chez Microsoft®, « TurboPlay » chez RealNetworks® et « Instant-on Streaming » chez Apple®.

1.4.1.2 Mise en cache durant la lecture

Comme nous venons de le voir, avant le démarrage de la restitution d'un stream, des données sont mises en cache. Ces données sont renouvelées durant la restitution du stream de manière à assurer une restitution non saccadée. Parmi les politiques de remplacement utilisées à ce niveau, on recense FIFO et LRU [BK75]. FIFO remplace les blocs de données dans l'ordre où ils y ont été insérés, *premier arrivé premier remplacé*, tandis que LRU se base sur la date de la dernière utilisation de blocs, *premier utilisé premier remplacé*. Étant donné que les données peuvent arriver, dans un réseau à commutation de paquets, de façon désordonnée, LRU est meilleure que FIFO pour la gestion de caches de lecture.

Les caches au niveau du client peuvent avoir d'autres objectifs, comme nous allons le voir au chapitre 3, où nous utilisons un cache de lecture afin d'améliorer les temps de réponse à des interactions de type avance rapide ou recul rapide.

1.4.2 Cache de serveur

Lorsqu'un serveur transmet des données, celles-ci sont momentanément stockées dans la mémoire, appelée cache de serveur, avant d'être expédiées. La problématique de la gestion du cache de serveur peut être divisée en deux tâches principales. La première concerne directement les politiques de gestion proprement dites, c'est-à-dire le remplacement de blocs. La deuxième s'intéresse aux opportunités d'optimisation de l'utilisation du cache. La frontière entre les deux tâches est souvent floue. En effet, les techniques proposées dans la littérature sont, dans leur majorité, des techniques dites intégrées, combinant à la fois les techniques de gestion et d'optimisation de l'utilisation des caches. Notons que l'objectif de toutes ces stratégies est de diminuer le taux d'échec du cache lors de la récupération de blocs de données et de réduire par conséquent le besoin de récupérer des données depuis le système de stockage du serveur.

Cette section passe en revue les techniques les plus représentatives dédiées à la gestion et à l'optimisation de l'utilisation d'un cache de serveur de streaming.

1.4.2.1 Stratégies de remplacement

L'objectif d'une stratégie de remplacement consiste à évincer, une fois la mémoire pleine, les blocs les moins propices au maintient dans le cache. Une telle stratégie est chargée de choisir, de manière aussi optimale que possible, les blocs qui seront victimes du remplacement, c'est-à-dire les blocs auxquels on aura accès le plus tardivement. Or, les techniques traditionnelles de remplacement, telles que LRU ou MRU ne prennent pas en compte les accès futurs aux données issues des streams en cours de diffusion et s'avèrent donc inadaptées à la gestion des caches de serveurs de streaming. En effet, lorsqu'il est nécessaire de remplacer un bloc, les techniques LRU et MRU se basent sur la date de la dernière utilisation des blocs pour effectuer ce remplacement. Le bloc à remplacer est celui auquel on a accédé le moins récemment pour LRU alors qu'il est celui auquel on a accédé le plus récemment pour MRU. Toutefois, puisque le débit, constant, d'un contenu audiovisuel est connu d'avance, il est possible de connaître *a priori* les blocs auxquels on aura accès ultérieurement pour les différents streams, grâce au fait que les accès aux contenus audiovisuels se font de manière séquentielle [6].

La stratégie BASIC [ÖRS96] exploite cette possibilité en prenant en compte la progression des différents streams. Elle assigne à chaque bloc de données d'un stream un *temps de référence*, calculé en fonction de la position du bloc dans le stream. Les blocs ayant le plus long temps de référence, c'est-à-dire les blocs auxquels on aura accès plus tardivement, sont sélectionnés comme blocs candidats au remplacement. Elle examine d'abord les blocs n'appartenant pas aux streams en cours de diffusion, puis le reste des blocs.

1.4.2.2 Stratégies d'optimisation

L'objectif recherché par une politique d'optimisation de l'utilisation de caches est de prendre en charge autant de streams que possible avec le même accès au système de stockage, augmentant ainsi le taux de succès du cache. Cette optimisation se traduit par un partage intelligent des données entre différents streams accédant aux mêmes contenus. Les techniques permettant d'effectuer ce partage, appelé aussi *patching* [7], sont basées sur l'idée de préserver momentanément des blocs de données dans la mémoire pour une réutilisation ultérieure par des streams diffusés à des instants relativement proches.

Le patching, tel qu'il est décrit par les deux politiques DISTANCE [ÖRS96] et GIC [DS97], repose sur la notion de *distance* entre les différents streams. Notons B_{ij}, pour marquer le bloc demandé par le stream j issu d'un contenu audiovisuel i (cf. Figure 1.18). Les blocs de données, qui se trouvent entre les deux blocs demandés par deux streams consécutifs j, $j+1$ accédant au même contenu audiovisuel i, sont appelés un *intervalle*. Ces deux streams, associés à un intervalle, sont appelés respectivement le *prédécesseur* et le *successeur*. La *taille de l'intervalle* est évaluée par le nombre de blocs qu'il comprend $(1 + B_{i(j+1)} - B_{ij})$. La distance d'un stream j issu d'un contenu audiovisuel i, notée d_{ij} est définie par la taille de l'intervalle dont il est prédécesseur. La distance d'un stream n'ayant pas de successeur est présumée infinie.

6. En faisant l'hypothèse qu'il n'y a pas d'interaction, de la part des utilisateurs, ayant pour effet de changer ce mode d'accès séquentiel propre aux données audiovisuelles.

7. À ne pas confondre avec le « batching », une technique d'ordonnancement de requêtes présentée dans la Section 1.3.3.3.

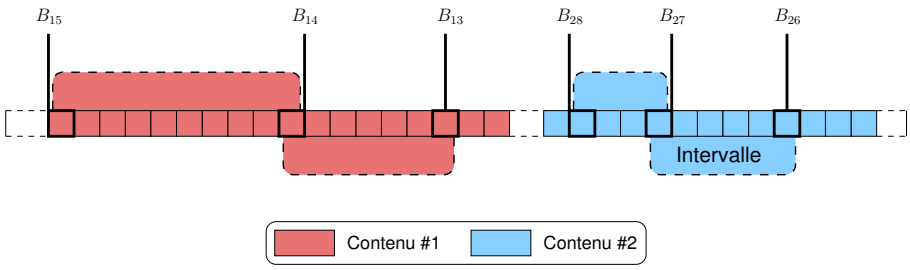


Figure 1.18 – Distance entre des streams du point de vue de la mémoire.

La technique DISTANCE classe les streams, à chaque cycle du serveur, selon l'ordre croissant de leurs distances. Ensuite elle remplace, si nécessaire, les blocs transmis dans le cycle précédant dans l'ordre décroissant des distances. La technique GIC travaille au niveau des intervalles et ne préserve les blocs issus de deux streams demandant l'accès au même contenu, associés à un intervalle, que s'ils étaient à une distance relativement petite l'un de l'autre. Elle classe les intervalles en fonction de leur taille et préserve uniquement les intervalles dont la taille est inférieure à un seuil, appelé *seuil de partage*. Le seuil de partage est un paramètre particulièrement important, compte tenu du volume des données audiovisuelles. Il doit être choisi de manière à assurer un bon équilibre entre la taille de la mémoire disponible et la distance de partage autorisée. GIC modifie le classement des intervalles uniquement lors du démarrage d'un nouveau stream ou lors de la fin d'un stream en cours de diffusion. Ceci la rend moins coûteuse que DISTANCE en terme de puissance de calcul.

1.4.3 Caches de réseau (« Proxy »)

Cette sorte de cache, appelé proxy, prend la forme d'un logiciel hébergé par une plate-forme matérielle qui lui permet d'exercer, à la fois, les fonctionnalités d'une passerelle entre les clients et le serveur, ainsi que celles d'un serveur ne possédant pas de contenu initial. Lorsqu'un contenu lui est demandé, il va, tout comme un serveur, transmettre ce contenu s'il le possède. Sinon, il agit comme une passerelle en déléguant la requête au serveur principal et en retransmettant le contenu au client demandeur, aussitôt que les données lui arrivent. Dans tous les cas, le proxy exerce un contrôle d'admission pour vérifier qu'il est en mesure d'honorer, ou non, la requête. Un proxy peut choisir de stocker (*cacher*), ou non, la totalité ou une partie du contenu, auquel on souhaite accéder au moment de la demande. Notons que l'espace de stockage géré par un proxy est largement inférieur au volume des contenus audiovisuels entreposés dans le serveur principal.

Les proxies sont placés stratégiquement, par des compagnies spécialisées dans la distribution de contenus (CDN) telles qu'Akamaï [aka], au plus près des usagers, au niveau des nœuds de raccordement d'abonnés (NRA) par exemple. Les proxies sont généralement connectés au serveur *via* des réseaux longue distance à haut débit. Les clients quant à eux sont connectés aux proxies *via* des réseaux grand public comme l'ADSL ou l'UMTS. La Figure 1.19 illustre ce modèle à trois niveaux : serveur-proxy-client. Cette architecture permet d'améliorer la performance d'un système de streaming et notamment la capacité du serveur :

- en transmettant un contenu répliqué depuis un proxy, la charge du serveur et le trafic entre le serveur et le proxy se trouvent donc réduits ;
- les temps de réponse aux clients et la probabilité de perte de paquets se voient diminués puisqu'un contenu transmis depuis un proxy parcourt un chemin plus court.

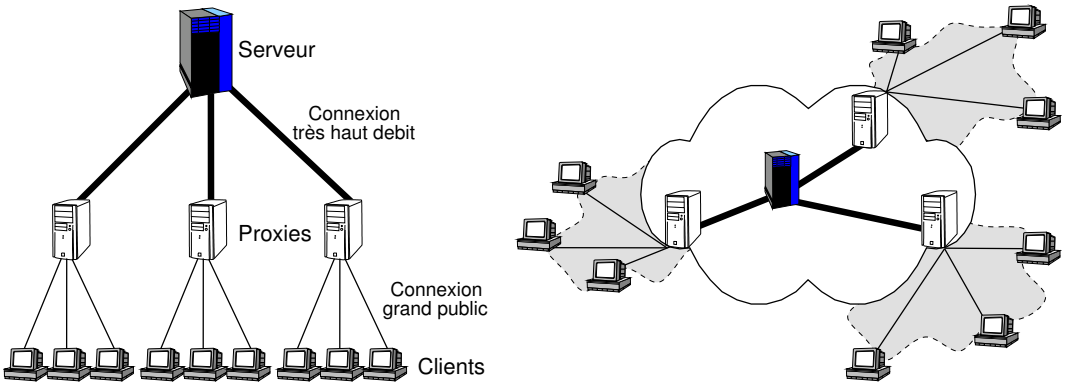


Figure 1.19 – Architecture à trois niveaux : serveur-proxy-client.

Les fonctionnalités des proxies ne se limitent pas à celles évoquées ci-dessus, elles peuvent inclure un traitement sur les données détenues, avant de les transmettre aux clients, comme par exemple l'adaptation des contenus au réseau sous-jacent. Dans la suite, nous présentons les techniques les plus couramment utilisées pour la gestion et la coopération entre des proxies.

1.4.3.1 Politiques de gestion de proxy

Compte tenu du volume colossal que représentent les données audiovisuelles, une mise en cache sélective des contenus disponibles dans le serveur est effectuée [MO02]. Cette mise en cache sélective passe par une prise de décisions concernant d'une part la suppression des contenus dont le stockage n'est plus utile, et concernant d'autre part l'admission de nouveaux contenus dans son système de stockage. Dans les paragraphes suivants, nous survolons les politiques permettant à un proxy de faire ces choix [LX04]. Ces politiques visent toutes à augmenter le taux de succès, qui correspond au pourcentage de blocs de données servis directement depuis le proxy, précisément depuis son espace de stockage ou depuis sa mémoire, par rapport au nombre total de blocs de données servis par le proxy.

Politiques d'insertion

Lorsqu'un proxy accepte de servir une requête, il vérifie la présence du contenu demandé dans son espace de stockage. Si le contenu n'est pas disponible localement, la politique d'insertion est interrogée afin de savoir s'il est judicieux de le stocker ou non, tout en le fournissant au client demandeur. Il existe de nombreuses politiques d'insertion dans la littérature. La plus connue est celle qui n'accepte de stocker le contenu que s'il a déjà été demandé [BH98]. D'autres se basent sur la fréquence des demandes. Ces dernières utilisent le même principe que la première, sauf que le nombre de demandes doit être supérieur à un seuil fixé pour que le contenu soit destiné à être stocké dans un cache. Bien entendu, si l'espace de stockage disponible ne suffit pas à effectuer ce stockage, le contenu sera stocké à la place d'un ou plusieurs autres, choisi(s) par la politique de suppression adoptée.

Politiques de suppression

L'espace de stockage d'un proxy étant limité, il lui arrive souvent d'avoir besoin de renouveler les contenus qui y sont conservés au cours du temps et donc de rester en phase avec les demandes des clients. Faute de quoi, le cache deviendra inutile car entièrement rempli de données qui ne seront plus utilisées. Afin de permettre ce renouvellement et libérer de l'espace de stockage, les contenus les moins propices au maintient sont évincés. Cette tâche est dévolue aux politiques de suppression. Celles-ci peuvent être décomposées en deux familles, selon le critère utilisé pour choisir le contenu à supprimer en premier, en cas de besoin d'espace de stockage.

La première famille, dont les membres les plus connus sont LRU et LFU, se base sur la popularité des contenus. La politique LRU [BK75] les classe par la date du dernier accès tandis que LFU [RD90] les classe par le nombre total d'accès depuis leur stockage. Ainsi, le contenu à supprimer en premier est celui qui a été utilisé le moins récemment pour LRU alors qu'il est celui qui est utilisé le moins fréquemment pour LFU. La politique LRU peut être trompée par un contenu peu demandé mais auquel on vient d'accéder. LFU peut être trompée, elle aussi, par un contenu très populaire à un moment donné mais dont la demande d'accès s'est fortement réduite.

La deuxième famille, dont le membre le plus connu est LFF, se base sur la taille des contenus. LFF [ASA+96] sélectionne le contenu le plus volumineux comme le candidat destiné à être supprimé en premier, libérant ainsi un maximum d'espace de stockage. D'ailleurs, en terme de gain d'espace, supprimer un unique élément volumineux est équivalant à supprimer plusieurs petits. Le fait de supprimer un seul contenu contenu diminue donc, dans l'absolu, la probabilité de suppression d'un contenu qui sera demandé dans un futur proche. Néanmoins, la limitation de cette politique est flagrante dans une situation où un contenu volumineux est aussi très populaire.

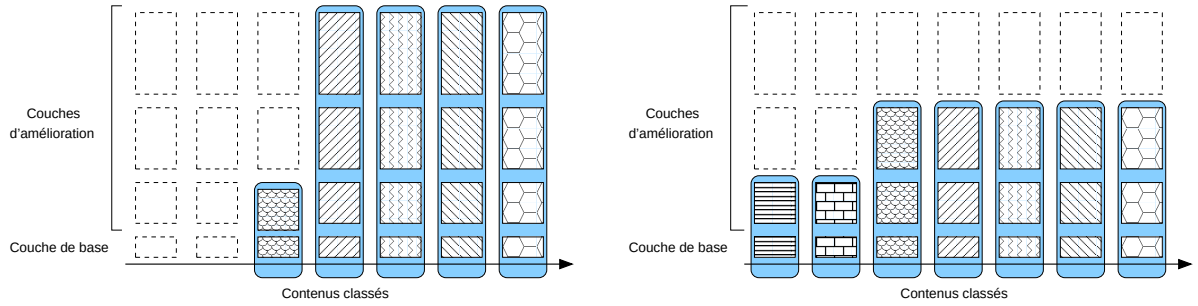
On recense dans la littérature de nombreuses variantes de ces politiques visant à combler leurs limitations [BK04]. À titre d'exemple, LRU-Seuil est basée sur LRU mais ne garde jamais en cache des contenus dépassant une certaine taille (Seuil). Une autre politique enlève le contenu qui a le plus grand $log(Taille)$ et qui a été demandé le moins récemment parmi tous ceux qui ont le même log(Taille). D'autres variations, telle que MIX [NLN98], Hybrid [WA97], ou encore les politiques les plus utilisées GDS et GDSF [BJ00, JB01a, JB01b], se basent sur un paramètre unique, souhaité optimal et appelé valeur de pertinence. Ce paramètre est issu d'une fusion de plusieurs paramètres. Par exemple, dans MIX, la valeur de pertinence d'un contenu est calculée suivant la formule :

$$\frac{(\text{Nb\_accès})^a}{(\text{Temps\_accès})^b \times (\text{Taille})^c} \tag{1.4}$$

où c, a, b sont des constantes permettant de pondérer, respectivement, la Taille du contenu, le nombre d'accès effectués depuis qu'il est en cache, Nb_accès, et la date du dernier accès à ce contenu, Temps_accès.

D'autres politiques de suppression, basées sur les précédentes, sont conçues spécialement pour des contenus encodés hiérarchiquement, en multi-couche (cf. Section 1.1.2.2). Dans ces politiques [KHRR02, PB02], la suppression est partielle et progressive. Les contenus, stockés dans le cache, sont d'abord classés par une des méthodes précédentes, LRU ou GDSF par exemple. Des couches sont ensuite progressivement supprimées, tant que l'espace de stockage libéré ne suffit pas pour accueillir un contenu jugé propice au stockage par la politique d'insertion adoptée. Cette

suppression de couches peut se passer de manière verticale ou horizontale (voir Figure 1.20). L'approche verticale supprime les contenus dans l'ordre croissant de leur popularité, mais en plusieurs étapes : si la politique s'arrête en n'ayant supprimé qu'une partie des couches d'un contenu, celui-ci sera toujours présent dans le cache, mais avec une qualité dégradée. Les meilleurs contenus sont intacts, les moins bons sont supprimés ou dégradés. L'approche horizontale tend plutôt vers un lissage de la qualité de l'ensemble des contenus. Un plus grand nombre de contenus est gardé en cache, mais la qualité globale se dégrade.



(a) Suppression verticale (b) Suppression horizontale

Figure 1.20 – Politiques de suppression optimisées pour des contenus encodés en multi-couche.

Mise en cache de préfixes

Cette technique, appelée aussi *prefix caching*, est employée comme un complément aux politiques d'insertion ou bien comme une politique d'insertion, afin de réduire le temps d'attente avant le démarrage de la restitution d'un contenu à un spectateur. Elle consiste à stocker dans le cache le début des contenus les plus demandés [SRT99]. Ainsi, lorsqu'une requête demande l'accès à un contenu dont le préfixe se trouve stocké dans le cache, le début du contenu est envoyé tout de suite au client pendant que la suite est demandée au serveur. L'espace de stockage est, quant à lui, divisé en deux zones distinctes : une zone dédiée aux préfixes et une zone de stockage usuelle gérée par une politique de gestion traditionnelle. Lorsque le mécanisme de suppression est déclenché, un contenu n'est pas simplement supprimé. Son préfixe est auparavant transvasé dans la zone dédiée. Si un nombre suffisant de requêtes pour ce contenu arrive, alors les clients pourront être servis de suite comme si le contenu était présent, et ce contenu sera à nouveau inséré dans la zone traditionnelle en rapatriant la portion supprimée. La zone des préfixes est un sas par lequel passent les contenus avant d'être supprimés. Il est à noter que la taille des préfixes joue directement sur la bonne marche du système.

Une généralisation de cette politique est donnée dans [WYW01], où un contenu est découpé en séries de *segments*. La taille d'un segment varie suivant la place qu'il occupe au sein du flux : plus le segment est éloigné du début du flux, plus il contient de blocs de données. Le nombre de blocs contenus dans le segment i est 2^{i-1}, sauf pour le premier segment numéroté 0 qui contient le premier bloc du stream. La Figure 1.21 illustre ce découpage en plusieurs segments. Durant la présence du contenu dans le cache, une valeur est assignée à chaque segment puis elle est mise à jour. Cette valeur est définie par le ratio entre la fréquence à laquelle le segment est demandé et sa distance temporelle $i + 1$. Les segments proches du début d'un flux audiovisuel sont ainsi privilégiés par rapport à ceux qui sont proches de la fin. L'effacement des segments se fait donc de la fin du stream vers le début.

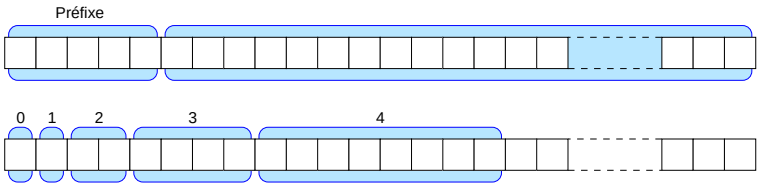


Figure 1.21 – Découpage d'un contenu audiovisuel en segments.

1.4.3.2 Proxies coopératifs

De toute évidence, l'efficacité de l'utilisation des proxies n'est garantie à 100% que si l'audience est équitablement répartie autour de ces proxies, de sorte que la majorité de la charge ne repose pas en un point local. Afin d'équilibrer la charge entre les différents proxies, des techniques permettant de les faire collaborer lors de l'acheminement de contenus aux clients ont été proposées. Ces approches exploitent la possibilité de distribuer un contenu audiovisuel sur plusieurs proxies. Elles utilisent ensuite des protocoles permettant la réception d'un contenu audiovisuel depuis plusieurs sources simultanément [NZ02]. Les données peuvent alors être transférées via plusieurs chemins en même temps, évitant ainsi une éventuelle congestion sur l'un d'eux (voir Figure 1.22).

L'utilisation de l'encodage par descriptions multiples permet de tirer un profit maximal de la coopération entre les différents proxies. Les différentes descriptions sont stockées sur des proxies différents de manière à ce qu'elles soient transmises simultanément à un client, chacune empruntant un chemin différent [AWWT02].

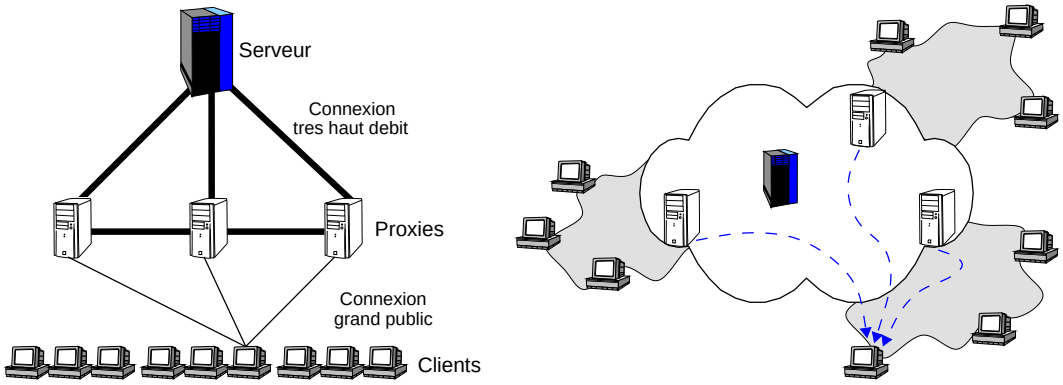


Figure 1.22 – Proxies coopérant pour acheminer les différentes descriptions d'un contenu audiovisuel aux clients via des chemins différents.

1.5 Streaming distribué

La capacité d'un serveur à servir simultanément une population importante ainsi que sa robustesse face aux diverses pannes sont intrinsèquement liées aux caractéristiques de la plate-forme matérielle l'hébergeant. Ainsi, les serveurs de streaming sont souvent optimisés pour un système d'exploitation et/ou une infrastructure matérielle donnés. Le coût a toujours été le facteur déterminant dans le choix de cette plate-forme.

Historiquement, les machines massivement parallèles (SMP), aussi appelées supercalculateurs, ont été les premières à héberger les serveurs de streaming [JSCB95]. Ces infrastructures onéreuses sont composées de plusieurs centaines, voire milliers, de processeurs homogènes et reliés entre eux par un réseau haute performance. Elles furent les seules infrastructure à posséder les ressources nécessaires à un serveur en termes de puissance de calcul, de taille mémoire, et de capacité de stockage. Parmi les serveurs développés au début des années 90, pour ces plates-formes, nous citons Fellini [MNO+96] de AT&T, commercialisé sous le nom de CineBlitz en 1997, ainsi que Oracle Media Server [LOP94] qui exploite l'immense capacité de la machine parallèle NCube et supporte jusqu'à 25000 streams simultanés.

Les machines massivement parallèles souffrent à la fois d'un rapport performance/prix nettement déséquilibré et d'une mauvaise *extensibilité*. Par extensibilité, appelée aussi capacité de *passer à l'échelle* (« scalability »), nous entendons la capacité à maintenir un temps de réponse constant lorsque le nombre de requêtes et/ou le nombre et la taille des contenus augmentent. En outre, un serveur, hébergé dans une machine massivement parallèle, est rendu vulnérable aux pannes à cause de l'unicité de la machine. Ainsi, à cause de leur coût élevé, les machines massivement parallèles ont été remplacées par des *machines virtuellement parallèles*.

Une machine parallèle virtuelle est obtenue en fédérant des machines peu chères et interconnectées via un réseau quelconque. Chaque machine, appelée nœud, est une machine standard (un ordinateur de bureau), équipée d'un système d'exploitation standard (souvent GNU/Linux) et possède sa propre mémoire et son propre espace de stockage. Les machines parallèles virtuelles constituent une solution aux limites architecturelles des SMP, notamment l'extensibilité et la tolérance aux pannes. En effet, l'emploi de machines parallèles virtuelles pour héberger un serveur de streaming contribue à le rendre à la fois rentable, tolérant aux pannes mais surtout « facilement » extensible (« scalable »). Les opérations d'extension d'une machine parallèle virtuelle se limitent à la connexion d'une nouvelle machine à l'ensemble existant, augmentant par conséquent ses ressources.

Ainsi, le serveur de streaming, en tant que logiciel, est distribué sur l'ensemble des nœuds constituant la machine parallèle virtuelle. On parle alors de *systèmes de streaming distribués*. Le serveur dans de tels systèmes est rendu encore plus complexe car l'ensemble des nœuds qui constituent une seule machine logique, doit être orchestré afin de le faire agir comme un seul serveur, grand et efficace.

Selon la localisation des nœuds et la qualité de l'interconnexion, on distingue deux catégories de machines virtuelles parallèles [FKT01]. Si les ordinateurs sont localisés sur le même site (une université ou une entreprise) et interconnectés *via* un réseau local caractérisé par une faible latence et une bande passante élevée, on parle alors d'une *grappe de machines* (« cluster »). Si les machines sont réparties géographiquement à travers le monde et interconnectées par des réseaux longue distance, on parle alors d'une *grille* (« grid »).

Nous présentons, dans cette section, une taxinomie des serveurs distribués en mettant en relief les points forts de chaque approche. Nous introduisons également une autre classe émergente de systèmes de streaming distribués, en l'occurrence les *systèmes de streaming collaboratifs* ou *systèmes de streaming pair-à-pair*.

1.5.1 Serveurs sur grappe de machines

Les serveurs sur grappe de machines, appelés aussi serveurs parallèles, se sont imposés comme un remplaçant rentable des serveurs sur des machines massivement parallèles grâce à l'évolution des réseaux d'interconnexion. En effet, en utilisant des normes telles que Ethernet ou Myrinet (voir Figure 1.11), on peut atteindre des débits de transfert allant jusqu'à 10 Gbits/s entre les différents nœuds et faire ainsi évoluer la performance d'une grappe vers celle d'une machine massivement parallèle équivalente en terme de nombre de processeurs. Toutefois, la performance d'un serveur sur grappe reste toujours tributaire de sa configuration. Une configuration naïve, où les différents nœuds sont indépendants et autonomes, présente deux inconvénients [Lee98] :

- La charge du serveur est distribuée de façon inégale entre les nœuds. En effet, les nœuds qui abritent les contenus les plus populaires sont surchargés tandis que d'autres restent sous-chargés.
- L'accès à un contenu très populaire est limité par la capacité du nœud le détenant. Afin de permettre à plus de spectateurs d'accéder au même contenu, ce dernier doit être copié sur d'autres nœuds. Or, ceci est inefficace à l'égard de la capacité de stockage du serveur. En outre, la variation de popularité des différents contenus rend la mise à jour des données populaires complexe.

Afin de pallier ces inconvénients, les données d'un contenu sont réparties sur plusieurs nœuds. L'accès aux différents contenus peut, lui aussi, être effectué via plusieurs nœuds. Les nœuds d'une grappe sont ainsi divisés logiquement en deux catégories [TJW03] : les noeuds d'interface (« front-end nodes » ou « access servers ») et les noeuds de stockage (« back-end nodes » ou « storage sub-system »).

Les noeuds d'interface assurent la communication avec les clients. Ils reçoivent les requêtes des usagers, les analysent (contrôle d'admission) et les délèguent éventuellement aux différents nœuds de stockage sur lesquels résident les données demandées. Les nœuds de stockage transmettent ensuite les données qu'ils détiennent aux clients, en respectant leur isochronisme. Que les données demandées transitent par les nœuds d'interface ou non influence significativement le comportement du serveur. De ce point de vue, on distingue deux architectures de serveurs parallèles : *architecture à deux tiers* et *architecture dite flat*. Il est à noter que dans la pratique, afin d'équilibrer la charge entre les différents nœuds d'interface, les deux architectures emploient un mécanisme pour orienter les requêtes vers les nœuds d'interface appropriés. Notons également que quelle que soit l'architecture adoptée au sein d'une grappe, les trois niveaux du modèle serveur-porxy-client sont toujours présents, mais on utilise un serveur parallèle.

1.5.1.1 Architecture à deux tiers

Dans cette architecture, les deux types logiques de nœuds sont bien distincts physiquement ; ils sont localisés sur des machines différentes. Les noeuds d'interface agissent comme des relais

entre les clients et les nœuds de stockage. Ces derniers reçoivent les requêtes des nœuds d'interface, récupèrent les données demandées et les transmettent aux nœuds d'interface qui, à leur tour, les retransmettent aux clients. Cette transmission, à deux étapes, provoque un taux élevé de trafic à l'intérieur de la grappe et entraîne, par conséquent, une dégradation importante de ses performances. Ainsi, une caractéristique marquante de cette approche est que la transmission de N octets se traduit par une transmission de $(2{\cdot}N)$ octets à l'intérieur du serveur. Aussi, les nœuds d'interface ne communiquent pas entre eux et la distribution des données sur les nœuds de stockage est assurée par un mécanisme de disques virtuellement partagés (voir Figure 1.23). Cette architecture peut employer les deux modèles d'interaction « client pull » mais aussi « serveur push ». Des exemples d'une telle architecture peuvent être trouvés dans [CT97, BVZ98], ainsi que dans le serveur baptisé Tiger [BBD+96] de Microsoft®.

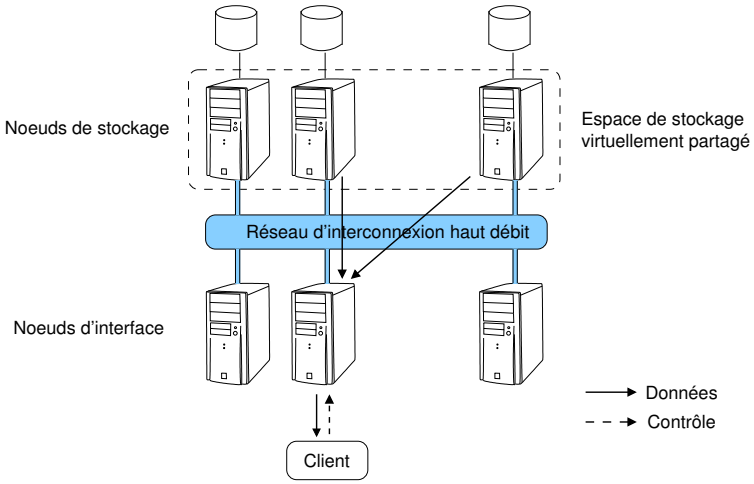


Figure 1.23 – Serveur en grappe, architecture à deux tiers.

1.5.1.2 Architecture flat

Par contraste avec l'approche précédente, chaque nœud physique constitue, dans cette architecture, à la fois un nœud d'interface et un nœud de stockage. Chaque nœud doit donc assurer simultanément les fonctions de communication d'un nœud d'interface et les fonctions d'un nœud de stockage (voir Figure 1.24). L'avantage de cette architecture est qu'elle génère moins de trafic de données à l'intérieur du serveur car il n'y a plus de phase supplémentaire de récupération des données avant de les expédier aux clients. Selon le modèle d'interaction adopté, les serveurs qui ont retenu cette architecture comme une base de conception, peuvent être divisés en deux catégories : architecture « client pull » et architecture « server push ».

Architecture « Client pull »

Un nœud, en tant que nœud d'interface, fournit la liste des nœuds qui détiennent les données demandées. Ainsi, il incombe au client de communiquer individuellement avec ces nœuds et d'assurer la synchronisation entre les différents streams issus de différents nœuds. Cette architecture a été tout d'abord proposée par Biersack en 1995 [BB95]. Les serveurs Calliope [HSE96] et SPIFFI [FD95] ainsi que les travaux de recherche de J. Lee [Lee05] ont adopté cette architecture.

Architecture « serveur push »

Cette architecture est proposée pour surmonter les lacunes de transparence de l'architecture « client pull ». Elle donne l'impression au client qu'il communique avec un seul nœud : le nœud d'interface. Le client envoie sa requête au nœud d'interface qui à son tour demande aux nœuds de stockage de lui transmettre les données sans intermédiaire. La synchronisation globale est effectuée par le nœud d'interface qui contrôle les différents streams issus des différents nœuds de stockage. De nombreux serveurs ont adopté cette architecture [TWJX03], nous citons : Yima-2 [SZFY02], SESAME-KB [BKB00], Odyssey [WSL03] et Elvira [SLM97].

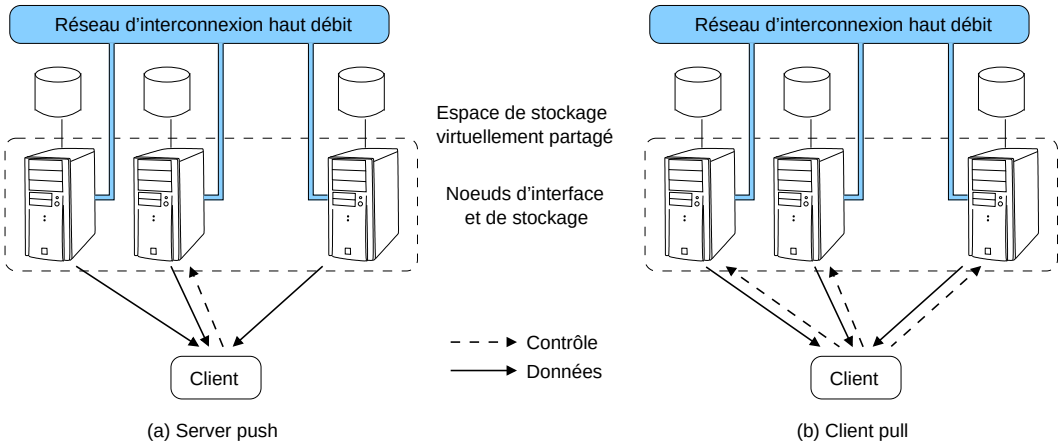


Figure 1.24 – Serveur en grappe, architecture flat.

1.5.2 Serveur sur grille

Une grille peut être constituée de l'agrégation de plusieurs machines distantes, de plusieurs machines massivement parallèles, de plusieurs grappes distantes (une grappe de grappes), voire d'un mélange des précédentes architectures. Ces formations transinstitutionnelles de machines, chacune appartenant à une institution différente, sont également appelées des « Organisations Virtuelles » [FKT01]. Une grille permet ainsi de fédérer un nombre beaucoup plus important de ressources qu'une grappe. Elle constitue donc une infrastructure alternative pour héberger les serveurs de streaming.

Les ressources d'une grille sont généralement stables, en dehors des pannes ou des opérations de maintenance relativement peu fréquentes, puisqu'elles lui sont dédiées. Toutefois, les machines d'une grille sont fortement hétérogènes tant au niveau matériel que logiciel. D'ailleurs, l'hétérogénéité d'une grille se propage jusqu'au réseau d'interconnexion reliant ses ressources, qui sont par définition, distantes. Afin de masquer cette hétérogénéité aux applications réparties (des applications destinées pour être exécutées sur grille), des logiciels médiateurs, appelés aussi intergiciels (« middlewares »), ont vu le jour. Les logiciels médiateurs peuvent être vus comme un environnement ou une couche logicielle permettant à des applications réparties d'être indépendantes des systèmes d'exploitation et des plates-formes matérielles sous-jacentes, et facilite par conséquent leur développement. En effet, les intergiciels offrent aux applications des services de base comme la gestion des ressources, les communications, le démarrage d'une activité, l'accès et le stockage des données, etc. Les principaux standards pour les logiciels médiateurs sont ODP-RM [FLdM95] de l'ISO et CORBA [GGM99] de l'OMG.

Les applications réparties font donc usage de logiciels médiateurs et les serveurs de streaming n'en font pas exception à un détail près : le streaming de données audiovisuelles impose des contraintes de temps réel strictes. Heureusement, la norme ODP-RM [GL97] supporte le temps réel, tout comme la norme CORBA [SK00] à partir de la version 2.5. Ainsi, de nombreuses plates-formes ont été proposées dans la littérature pour supporter le streaming audiovisuel sur grille [MBK+97, LZSG03, ZBP04]. Nous citons par exemple TAO [MSSK99], MAESTRO [HSK+99] et Polka [FGD96].

Il est important de noter que les nœuds d'un serveur sur grille peuvent être vus comme des proxies. Ainsi, les deux niveaux, serveur et porxy, du modèle serveur-porxy-client sont réunis en un seul niveau incarné par les nœuds de la grille. Cette fusion est un pas vers la fusion totale des trois niveaux en un seul et unique niveau, donnant un système de streaming dit *système de streaming pair à pair*.

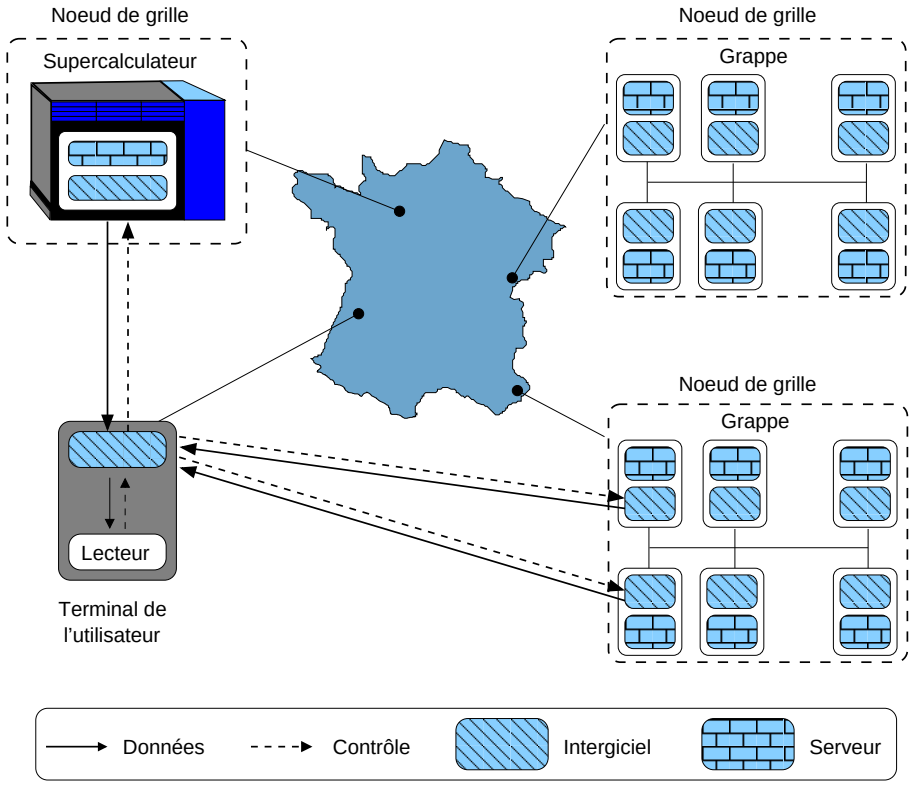


Figure 1.25 – Serveur en grille.

1.5.3 Streaming collaboratif

Avec la démocratisation des terminaux dotés d'une grande puissance de traitement et disposant de connexions à débits importants, il est désormais possible que les clients hébergés sur ces terminaux s'attellent à la tâche de streaming, autrefois strictement réservée aux serveurs. On parle alors de streaming collaboratif ou systèmes du streaming pair-à-pair (P2P). Cette nouvelle facette émergente du streaming distribué, où les clients, appelés dorénavant *pairs*, peuvent eux aussi jouer le rôle de serveur, est particulièrement intéressante. En effet, les ressources, réparties

à travers le monde et capables de prendre en charge les tâches d'un serveur de streaming, deviennent virtuellement illimitées. Toutefois, ces ressources sont très volatiles dans le sens où les pairs peuvent rejoindre ou quitter le système à n'importe quel moment et en particulier durant le streaming d'un contenu. D'ailleurs, ces ressources sont très hétérogènes et interconnectées via des réseaux hétérogènes et aux débits variés. La volatilité des systèmes de streaming pair-à-pair entraîne un problème de disponibilité des contenus au sein du système. Ce problème est étudié en détails au Chapitre 3, qui est consacré aux systèmes de streaming P2P.

* * *

Chapitre 2

Interactivité dans les présentations multimédias

Dans ce chapitre, nous considérons le streaming de documents multimédias au sens large du terme. Un document multimédia est le résultat d'une composition organisée d'un ensemble d'objets médias de différentes natures comme du texte, des images, du son et de la vidéo. Cette composition prend des dimensions spatiales et temporelles *via* des relations définies entre les objets faisant partie du document multimédia. La présentation d'un tel document consiste à restituer les objets le composant en respectant d'une part les contraintes imposées par les relations de composition entre ces objets et d'autre part les contraintes temps réel des objets audiovisuels qu'il contient.

De nos jours, la notion de *présentation multimédia* est devenue très répandue et utilisée dans un spectre applicatif très large. En effet, son domaine d'application s'étend des applications professionnelles, telles que les bibliothèques virtuelles, aux applications grand public, telles que l'enseignement à distance en passant par les présentations commerciales et publicitaires. Cependant, et malgré cette démocratisation, les fonctionnalités permettant de les parcourir rapidement sont absentes des systèmes de streaming actuels. Cette absence est due aux difficultés, aussi bien techniques que sémantiques, qui sont rencontrées lors de la mise en œuvre de ce parcours rapide dans un environnement de streaming. Il s'agit principalement de la surconsommation de ressources réseau lorsqu'une présentation est accélérée de manière « conventionnelle ». Plus important encore, la présentation peut devenir inintelligible lors d'une telle accélération. L'objectif de ce chapitre est de décrire la contribution que nous apportons afin de surmonter ces obstacles, aussi bien sur le plan conceptuel que pratique.

Nous avons commencé par la caractérisation des interactions permettant une navigation accélérée dans les présentations multimédias. Ce travail nous a conduit à nous intéresser de plus près à la sémantique sous-jacente de ces interactions. Nous avons alors défini une sémantique adéquate de la navigation accélérée dans les présentations multimédias. L'originalité de notre approche réside dans le fait qu'elle relie le parcours rapide d'une présentation à son contenu. Ensuite, nous avons proposé d'utiliser le cache de lecture afin de faciliter la mise en œuvre de la sémantique proposée. Nous avons élaboré une politique de gestion de ce cache spécialement adaptée à la nouvelle sémantique du parcours rapide. Cette politique permet de réduire les temps de latence traditionnellement observés dans les systèmes de streaming.

Nous commençons par une introduction des présentations multimédias, de leur composition, des relations spatio-temporelles qui régissent leur restitution, de la synchronisation omniprésente lors d'une restitution et des types de navigation dans ces présentations. Nous poursuivons par une description de la nouvelle sémantique que nous proposons pour le parcours rapide des présentations multimédias. La politique de gestion du cache de lecture que nous avons mise au point pour permettre l'implantation de cette nouvelle sémantique de navigation accélérée est ensuite présentée. Nous clôturons ce chapitre par une série d'expérimentations et de simulations validant nos propositions.

2.1 Présentations multimédias

Une multitude d'objets médias, de différents types, peut être réunie, acheminée, et présentée ensemble selon un scénario spécifié ; on parle alors de *documents multimédias* ou de *présentations multimédias*. Un document multimédia est donc un regroupement « organisé » d'objets médias qui traitent d'un sujet précis ou d'idées fortement liées. Lorsqu'ils sont présentés, ces objets sont synchronisés afin d'obéir au scénario spécifié. Ainsi, nous distinguons deux dimensions temporelles : celle intrinsèque aux objets continus inclus dans une présentation et celle dictée par son *scénario*, lequel décrit l'enchaînement des objets dans le temps.

Dans cette section, nous synthétisons les notions principales liées aux présentations multimédias. Nous commençons par mettre l'accent sur les différents aspects caractérisant celles-ci, à savoir la composition et la synchronisation omniprésente durant une présentation. Ensuite, nous passons en revue les différentes approches de spécification des présentations multimédias. Nous enchaînons par une description des techniques utilisées afin d'adapter les systèmes de streaming aux présentations multimédias. Nous clôturons cette section par une discussion sur les différents types de navigation dans ces documents.

2.1.1 Composition

La présentation d'un document multimédia n'est pas une simple présentation d'objets médias indépendants. C'est un processus qui traite une multitude d'objets médias de natures hétérogènes, mais organisés logiquement, spatialement et temporellement. En effet, lorsqu'une présentation est rédigée par un auteur, ce dernier spécifie l'organisation logique de cette présentation, détermine le placement spatial des médias à perception visuelle et définit son scénario temporel. Nous illustrons ces trois dimensions que prend l'organisation d'un document multimédia à l'aide d'un exemple qui nous servira de point de repère lors de la description de différentes notions. Cet exemple est constitué d'un cours de chimie issu d'un établissement spécialisé dans l'enseignement à distance.

2.1.1.1 Organisation logique

Dans une présentation multimédia, il est possible de distinguer des parties liées sémantiquement. L'organisation logique d'une présentation consiste à regrouper ces parties par le biais de relations logiques. En d'autres termes, elle correspond à la décomposition de la présentation. L'opération de décomposition se termine lorsque nous arrivons à décomposer la présentation en objets médias de base (voir Figure 2.1). Le résultat de cette opération donne une structure logique hiérarchique de la présentation. L'organisation logique d'une présentation passe donc par la spécification de relations d'inclusion entre ses parties logiques.

La Figure 2.1 schématise l'organisation logique d'un cours de chimie composé de quatre parties : une introduction, des préliminaires, une partie regroupant les différentes notions mises en lumière dans le cours et une conclusion. La présentation de chaque notion est elle-même décomposée en une partie théorique et une partie pratique. La partie théorique peut être illustrée par des formules et des schémas sous forme d'images. La partie pratique peut illustrer une réaction par le biais d'une animation. Le discours du professeur, omniprésent dans le cours, est toujours accompagné d'une transcription textuelle pour les personnes malentendantes.

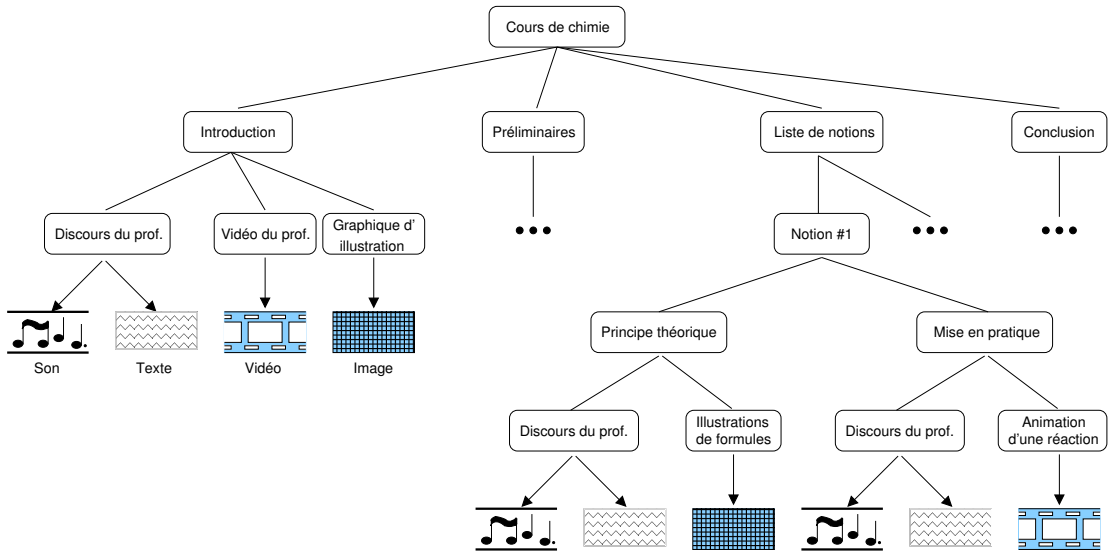


Figure 2.1 – Structure logique d'une présentation.

2.1.1.2 Organisation spatiale

L'organisation spatiale consiste à définir le placement, pendant la présentation, des objets médias à perception visible les uns par rapport aux autres sur un support de projection tel qu'un écran. Cela correspond à l'affectation d'espaces géométriques, à deux dimensions, à ces médias. À chaque objet à perception visible est associée une zone délimitant son affichage. Cette dimension spatiale d'une présentation est étroitement liée à sa dimension temporelle. En effet, une zone donnée peut être utilisée pour afficher plusieurs médias, chacun pendant une période distincte. Les périodes pendant lesquelles différents objets sont affichés sur une zone donnée ne se chevauchent pas. La Figure 2.2 matérialise une organisation spatiale possible du cours de chimie que nous avons pris comme exemple illustratif.

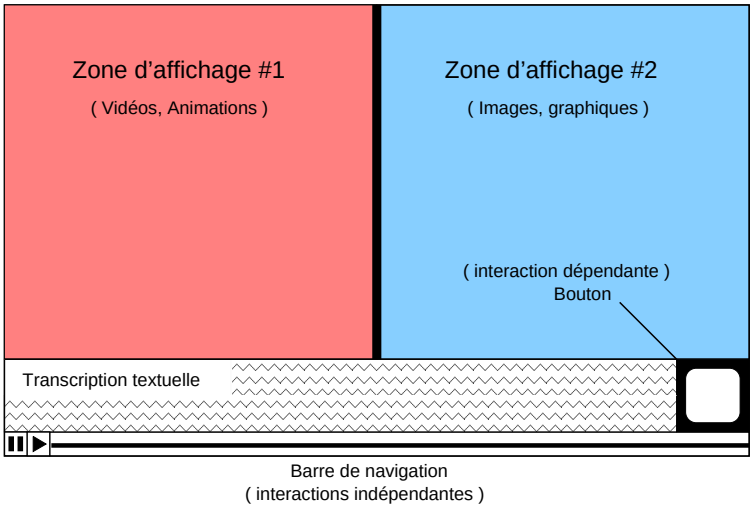


Figure 2.2 – Disposition spatiale d'une présentation.

2.1.1.3 Organisation temporelle

L'organisation temporelle, appelée aussi *scénario*, est l'élément principal caractérisant une présentation multimédia. En effet, le scénario d'une présentation est l'élément qui lui donne son aspect dynamique en spécifiant, par rapport à un axe du temps, les instants de démarrage et d'arrêt de restitution des objets constituant la présentation. Notons que ces instants peuvent être liés à des interactions faites par l'utilisateur. Le scénario spécifie également les contraintes de synchronisation entre les différents objets.

La Figure 2.3 présente le scénario correspondant à notre exemple. Elle montre l'ordre temporel de la présentation des différents objets constituant ce cours. On y voit que la vidéo du professeur, vidéo #1, est à restituer entre T_1 et T_2. D'ailleurs, cette vidéo doit être synchronisée avec son discours #1 et la transcription #1 de ce dernier.

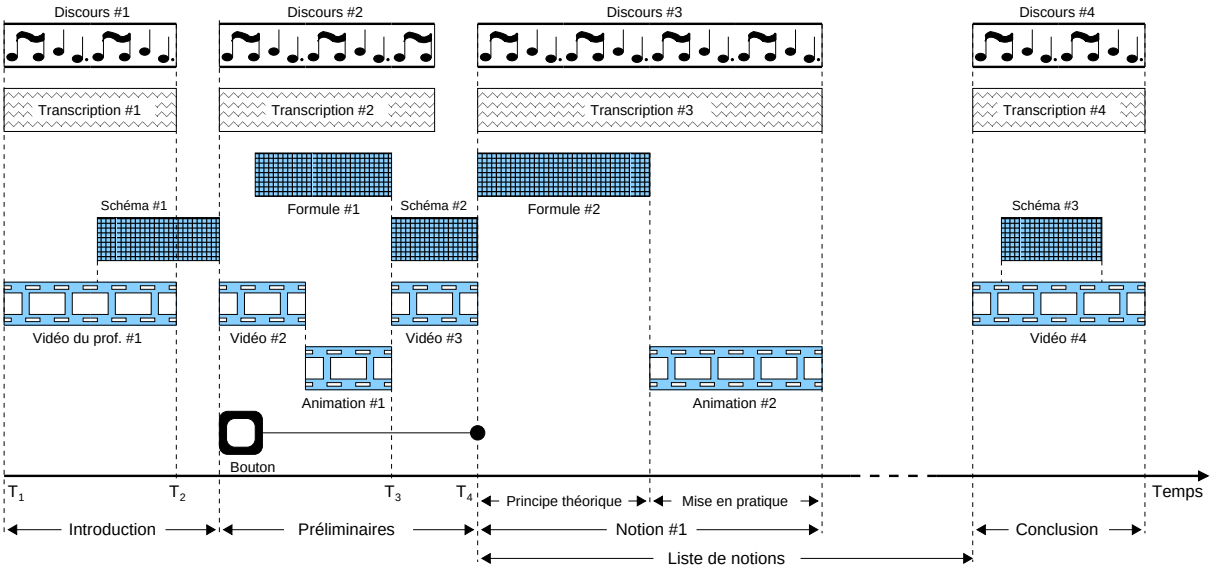


Figure 2.3 – Scénario temporel d'une présentation.

Le respect du scénario spécifié par l'auteur d'une présentation est primordial dans la mesure où il est indispensable pour une restitution compréhensive du document multimédia. Ainsi, en fonction du scénario spécifié, les ressources requises pour la restitution de différents objets sont allouées, mixées dans le cas du son, et libérées au fur et à mesure de l'avancement dans la présentation du document multimédia [OTTH92].

2.1.2 Synchronisation durant une présentation

La dimension temporelle apparaît dans une présentation multimédia de façon interne à ses objets continus ainsi que de façon externe aux objets à travers des relations temporelles définies par son scénario. La synchronisation apparaît quant à elle à différents niveaux de granularité : à l'intérieur des objets continus mais aussi entre les différents objets. La synchronisation intrinsèque aux objets continus, appelée aussi *synchronisation intra-objet*, est issue de l'isochronisme qui leur est inhérent (voir Section 1.1.1.1). Quant à la synchronisation entre des objets, on distingue les deux niveaux de granularité décrits ci-après.

2.1.2.1 Synchronisation à gros grain

Ce type de synchronisation, baptisé *synchronisation synthétique* ou *synchronisation inter-objets*, est spécifié entre différents objets médias. Elle se présente sous la forme de relations temporelles entre les instants de début et/ou les instants de fin de différents objets médias. Par exemple, dans le scénario illustré par la Figure 2.3, le schéma #2 s'affiche en même temps que commence la séquence vidéo #3 et reste affiché pendant $(T_4 - T_3)$ secondes. Ce schéma est suivi par la formules #2.

2.1.2.2 Synchronisation à grain fin

Ce type de synchronisation, dénommé *synchronisation naturelle* ou *synchronisation de lèvres* (« lip-sync »), impose un couplage temporel entre la progression temporelle de deux ou plusieurs objets continus. Dans le scénario présenté par la Figure 2.3, la séquence vidéo #1, le discours audio #1 et le texte #1 le transcrivant sont *synchronisés à grain fin*. Supposons que la vidéo et l'audio sont restitués à des cadences respectives de 25 images et 8000 échantillons par seconde, la synchronisation à grain fin entre ces deux objets se traduit par l'affichage d'une image vidéo tous les 320 échantillons audio (voir Figure 2.4). Le flux textuel est lui aussi synchronisé à grain fin avec le discours audio de manière à assurer une cohérence entre l'état d'avancement du discours et sa transcription textuelle. La synchronisation naturelle peut être trouvée dans bien d'autres situations : lorsqu'un carré est utilisé pour annoter un élément en mouvement dans une séquence vidéo par exemple.

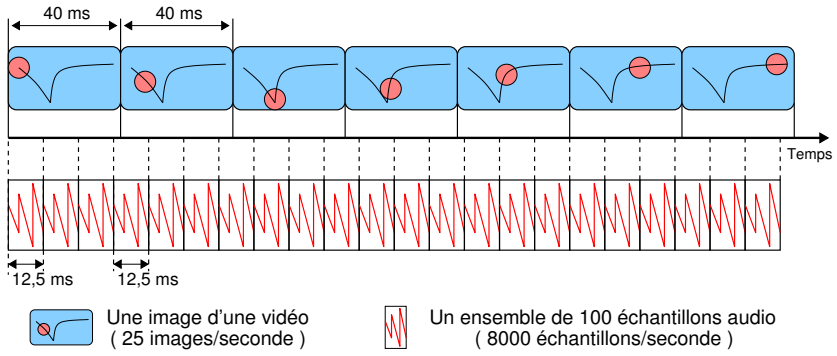


Figure 2.4 – Synchronisation à grain fin entre une vidéo et une piste audio.

2.1.3 Spécifications et environnements d'édition

Le standard XML [BPM+06] permet de représenter la dimension logique dans les documents, indépendamment de leur contenu et de leur dimension spatiale. Or la dimension temporelle, omniprésente dans les documents multimédias, nécessite d'avoir un formalisme pour décrire les scénarios temporels. Depuis plusieurs années, un axe de recherche très important porte sur la spécification de ces scénarios. En effet, de nombreuses approches ont été proposées afin de permettre la spécification spatio-temporelle et la synchronisation entre les différents objets faisant partie d'une présentation. Par spécification spatio-temporelles des objets, nous entendons la description du moment où un objet sera restitué, pendant combien de temps, ainsi que dans quelle partie de la fenêtre d'affichage (pour les objets à perception visible).

Quelle que soit l'approche de spécification des documents multimédias, un environnement d'édition lui est associé afin de permettre la composition de documents respectant la méthode de spécification choisie. Ces environnements permettent la composition de présentations multimédias à partir d'objets de base, considérés comme des atomes tels que des vidéos, des bandes son, des images ou des textes. De manière générale, ces outils d'édition permettent aussi la présentation de documents multimédias. Ce passage est particulièrement important car il permet à un auteur de percevoir le résultat de sa spécification et de l'éditer à nouveau s'il le souhaite.

Les approches de spécification peuvent être classées en quatre grandes familles décrites ci-après. Chacune de ces famille est caractérisée par sa puissance expressive permettant de spécifier des scénarios plus ou moins complexes et par la facilité de spécifier de tels scénarios.

2.1.3.1 Approches à base d'axe temporel absolu

Les approches fondées sur un axe temporel absolu (« timeline ») sont les plus répandues dans les outils commerciaux comme Adobe® Director®. De même, le standard HyTime [NKN91] est un langage qui s'appuie aussi sur la modélisation du temps par un axe. Notons qu'il existe un nombre très faible de réalisations d'outils d'édition de documents HyTime.

Ces approches sont accessibles aux non informaticiens et rendent compte, de façon intuitive, du placement temporel des différents événements de la présentation. En revanche, la représentation logique du document est aplatie, voire inexistante.

2.1.3.2 Approches à base de scripts

Les outils à base d'axe temporel, du fait de leur pouvoir d'expression limité, sont le plus souvent complétés par des langages de programmation, appelés « scripts ». C'est le cas d'Adobe® Director® dont le langage de script associé s'appelle Lingo. De même, le standard MHEG [MBE95] s'appuie sur le paradigme des langages de programmation orientés objet. MHEG est à l'origine de plusieurs prototypes tels que GLASS [GBB+97], réalisé en Java, et MAJA [BGH96], réalisé en Java et CORBA.

Il est clair qu'avec un langage de programmation, il est possible d'exprimer des scénarios aussi complexes que l'on veut. Le scénario est spécifié sous forme opératoire par un ensemble d'instructions « événement(s) & condition(s) → action(s) ». En revanche, les compétences requises pour utiliser les scripts restreignent leur emploi aux informaticiens. D'ailleurs, il est difficile de percevoir l'enchaînement par la simple lecture d'un script.

2.1.3.3 Approches à base de graphes

Dans les approches fondées sur les graphes, un document est représenté sous la forme d'un graphe de son flot de contrôle. L'édition de documents peut ainsi s'appuyer sur une description plus riche et plus facile à utiliser. En effet, en utilisant les graphes, il est possible de représenter beaucoup plus de combinaisons de scénarios que dans le cas des approches basées sur un axe temporel. Aussi, les graphes fournissent un support plus adapté aux opérations d'édition comme la suppression ou l'insertion d'objets. Il existe deux variantes de ces diagrammes : les graphes plats et les graphes hiérarchisés.

Graphes plats

Les graphes plats décrivent la synchronisation entre les instants de début et de fin de l'ensemble des objets d'une présentation. Souvent la synchronisation est décrite au moyen de trois opérateurs d'instants : « avant », « après » et « en même temps ». Ces graphes sont utilisés de deux façons. Dans Firefly [BZ92], leur utilisation se limite à un cadre informel de synchronisation, motivé par des besoins d'interface graphique. Dans HTSPN [SDLS96], cette utilisation correspond à un véritable langage pour des modèles temporels plus élaborés (réseaux de Pétri).

Bien que cette approche soit simple à utiliser, la description d'un document sous forme de graphe plat a tendance à devenir rapidement complexe et très peu lisible dès que sa taille croît. En outre, la représentation du document reste très peu structurée.

Graphes hiérarchisés

L'approche basée sur une structure de graphes hiérarchisés consiste à exploiter l'organisation logique du document pour décrire et mettre en œuvre sa synchronisation temporelle. Les inconvénients cités pour les graphes plats sont compensés dans les graphes hiérarchisés par des mécanismes d'édition plus élaborés, permettant d'encapsuler des parties du document et de les utiliser comme des objets de base. Cette approche est adoptée par CMIFed [RJMB93] ainsi que par le standard SMIL [BGJ+05], qui s'appuient sur une structure d'arbre d'opérateurs temporels. De nombreux outils ont été créés pour le traitement de documents respectant le standard SMIL, tels que GRiNS [BHJ+98], HPAS [YX97, YY98] et RealSystem G2™. Le système GRiNS, réalisé en Python, se charge de l'édition et de la présentation de documents SMIL alors que le système HPAS, réalisé en Java, se limite à leur présentation. Les deux systèmes accèdent aux objets médias en utilisant le protocole HTTP, tandis que RealSystem G2™ utilise les protocoles temps réel RTP, RTCP et RTSP.

2.1.3.4 Approches à base de contraintes

Ces approches s'appuient sur l'algèbre d'Allen [All83] pour exprimer la synchronisation temporelle au sein d'une présentation multimédia sous forme de contraintes temporelles. L'auteur d'une présentation déclare ce qu'il souhaite obtenir comme placement temporel d'objets sans à avoir à spécifier toutes les informations temporelles attachées à ces objets. Un scénario est donc défini par un ensemble de relations temporelles du type « avant », « pendant », etc. sans que les instants de début et de fin de chaque objet soient spécifiés. À partir de ces contraintes, les systèmes d'édition basés sur cette approche ont pour tâches de s'assurer d'abord de la cohérence du document, c'est-à-dire de l'existence d'au moins une solution qui respecte toutes les contraintes, et ensuite de produire statiquement ou dynamiquement une ou des solutions de placement temporel, pour les documents cohérents. Par analogie aux systèmes de mise en forme conventionnels, tels que LaTeX, on parle de *mise en forme temporelle*.

Cette approche facilite la modification de documents, puisque l'ajout ou la suppression d'un objet se traduit, respectivement, par l'ajout ou la suppression des contraintes le concernant. Ces modifications se répercutent automatiquement sur le reste du document lors de sa mise en forme. Cette approche est adoptée par de nombreux systèmes tels que ISIS [KS95], Tiempo [Wir97] et Madeus [JLR+98]. Afin de faciliter la présentation de documents spécifiés avec des contraintes, il est envisageable de les sauvegarder, lorsque leur forme est jugée suffisamment stable, dans des formats comme SMIL ou MHEG.

2.1.4 Systèmes de streaming multimédia

De par la complexité des présentations multimédias, des systèmes de streaming adaptés doivent être conçus afin d'assurer une restitution cohérente, respectueuse du scénario associé, et sans saccade de présentations multimédias, on parle alors de *systèmes de streaming multimédia*. Cette adaptation se répercute à tous les niveaux d'un système de streaming, c'est-à-dire serveur, réseau et client.

2.1.4.1 Serveurs multimédias

On recense dans la littérature deux améliorations des serveurs de streaming pour les adapter au streaming de présentations multimédias : l'une est architecturale et concerne la conception du serveur, et l'autre est fonctionnelle et concerne l'acheminement de présentations « adaptées » au réseau de transmission sous-jacent.

Approche de séparation versus approche d'intégration

Dans une présentation multimédia, deux classes de médias sont distinguées par leurs natures intrinsèquement différentes : les médias statiques et les médias dynamiques. Ces deux types de média nécessitent l'emploi de techniques spécifiquement adaptées lorsqu'ils sont transmis en streaming. Deux approches de conception de serveurs multimédias sont ainsi envisageables.

Approche de séparation Dans cette approche, le serveur multimédia est constitué de deux sous-serveurs distincts, chacun hébergé par une machine différente. Un premier sous-serveur est dédié aux médias statiques. Toute technique issue du domaine des bases de données classiques peut être utilisée pour la gestion de celui-ci. Un deuxième sous-serveur est dédié aux médias dynamiques : le serveur de streaming. Les techniques décrites dans la Section 1.3.3 peuvent être utilisées pour gérer ce sous-serveur. Cette approche offre l'avantage d'une implémentation relativement peu complexe et de meilleures performances puisque les techniques de gestion employées sont adaptées à chaque classe de média.

Approche d'intégration À l'inverse de l'approche précédente, les deux classes de média coexistent dans un seul serveur dit *intégré*. Ce dernier doit assurer de façon interne une gestion appropriée à chaque classe de média. Cette approche offre l'avantage d'une utilisation rationnelle des ressources mises à la disposition du serveur. En effet, lors d'une forte charge d'accès sur une classe de média, le serveur est en mesure de mobiliser la totalité de ses ressources, alors que cela n'est pas possible dans une approche de séparation. À titre d'exemple, le serveur Symphony [SGRV98], développé par l'Université du Texas, adopte cette approche.

Adaptation des présentations multimédias

L'accès en streaming aux présentations multimédias est rendu problématique, d'une part en raison de la multitude des objets, notamment les objets continus qui doivent être acheminés, décodés et restitués dans le respect de leur isochronisme ; et d'autre part, à cause de l'hétérogénéité des réseaux utilisés lors de la transmission. Le streaming adaptatif (cf. Section 1.3.3.6), compte tenu de la multitude d'objets que peut comprendre un document multimédia, prend toute son ampleur dans ce contexte.

En effet, l'adaptation dynamique de médias par dégradation de leur qualité, en utilisant les techniques de transcodage ou d'encodage adaptatif, trouve toute son utilité dans le contexte de streaming de présentations multimédias. D'ailleurs, le transmodage de certains objets d'une présentation s'avère particulièrement intéressant. Il s'agit de générer, de manière automatique, des versions d'une présentation par le biais de scénarios alternatifs. Chaque version est adaptée à un accès via une connexion d'une capacité donnée. Les scénarios alternatifs obéissent à des règles qui vérifient leur cohérence, et notamment le fait qu'ils sont toujours compréhensibles et conservent les mêmes informations que la présentation originale. Par exemple, une vidéo ne peut pas être remplacée par un discours descriptif si elle est déjà synchronisée avec un autre discours, elle peut dans ce cas être remplacée par une suite d'images qui sont des extraits de cette vidéo.

L'adaptation des présentations multimédias par transmodage de ses objets via des scénarios alternatifs a fait l'objet de nombreux travaux de recherche visant à créer des modèles permettant cette adaptation. Parmi ces modèles, nous citons AHM [HBvR94], XYZ [BK98], InfoPyramid [SML99], NAC [LLQ05] et le standard MPEG-21 [BdWH+03].

2.1.4.2 Optimisation de transmission

Au regard des caractéristiques hétérogènes des objets médias, des protocoles de transport adaptés à chaque type d'objets doivent être employés lorsqu'ils sont transmis. Des protocoles tels que HTTP ou FTP peuvent être utilisés pour la transmission de médias statiques tandis que la transmission de médias continus doit être effectuée par des protocoles dédiés, tels que RTSP (cf. Section 1.3.2.2).

Par ailleurs, il est tout à fait envisageable que des objets, ou des portions d'objets, faisant partie d'une même présentation soient répartis entre plusieurs proxies. Il en résulte une meilleure accessibilité aux différents objets acheminés *via* des routes différentes.

2.1.4.3 Lecteur de présentations multimédias

Dans cette section, nous donnons un aperçu d'un lecteur de présentations multimédias, faisant abstraction de toutes les spécifications de celles-ci. Néanmoins, nous supposons que le paradigme d'accès aux données est de type « demande par le client » (cf. Section 1.3.4). En effet, il est mieux adapté à l'accès aux différents objets se trouvant sur différents proxies et/ou serveurs.

Un lecteur de présentations permet de restituer les différent objets la constituant tout en respectant son scénario. On peut y distinguer quatre composants : le contrôleur, le gestionnaire d'accès, le gestionnaire d'objets, le gestionnaire de cache (voir Figure 2.5). À chacun de ces composants incombe des tâches spécifiques que nous décrirons dans les paragraphes suivants.

Contrôleur Le contrôleur est chargé d'ordonnancer la restitution des objets selon le scénario spécifié. Il assure la synchronisation entre les objets, quelle que soit sa granularité. Il s'occupe également de répondre aux désirs des usagers. Ces désirs sont exprimés par diverses interactions que nous détaillons dans la section suivante.

Lorsqu'un utilisateur demande l'accès à une présentation donnée, le contrôleur demande d'abord au gestionnaire d'accès de récupérer son scénario. Ensuite, à partir de ce scénario, le

contrôleur demande la récupération et la restitution des objets tout en s'assurant du respect des relations spatio-temporelles et de la synchronisation entre les objets.

Gestionnaire d'accès Le gestionnaire d'accès est chargé de récupérer le scénario de la présentation ainsi que les objets la constituant, qu'ils soient locaux ou à distance. La livraison des différents blocs de données est soumise à des contraintes en matière de délais maximaux autorisés par le contrôleur. Ainsi, lors d'un accès à distance, ce gestionnaire emploie des protocoles adaptés à la transmission de chaque type d'objets. Les blocs de données reçus sont stockés provisoirement dans un cache de lecture. Il est à noter que le gestionnaire d'accès communique avec le gestionnaire de cache afin de savoir si un bloc de données est déjà disponible dans le cache ou non. Si le bloc n'est pas disponible, il se charge alors de le récupérer.

Gestionnaire d'objets Ce composant est chargé de décoder et de restituer les différents objets au spectateur. Il contient des modules spécifiques au décodage et à la restitution de chaque type d'objets. Il est ainsi chargé de la synchronisation intra-objets. La restitution des différents objets est supervisée par le contrôleur, afin de respecter les relations spatio-temporelles et la synchronisation spécifiées par le scénario.

Gestionnaire de cache Les blocs de données, avant d'être restitués, transitent par un cache de lecture. La gestion de ce dernier incombe à ce composant. Cette gestion passe par l'emploi de politiques de remplacement de blocs de données lorsque le cache est plein.

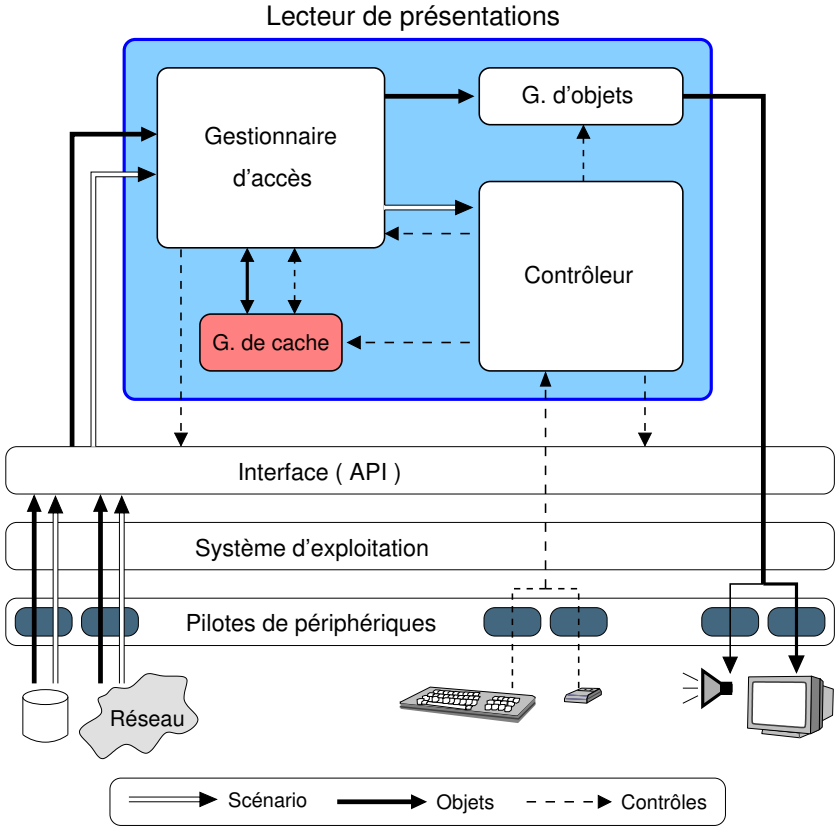


Figure 2.5 – Lecteur de présentations multimédias.

2.1.5 Navigation dans les présentations multimédias

On distingue deux types de navigation dans les présentations multimédias [LR04] : la navigation dépendante du document et la navigation indépendante du document. La différence majeure entre ces types de navigation réside dans le fait que la navigation indépendante n'a pas d'effet sur le scénario de la présentation, tandis que la navigation dépendante peut l'altérer.

Quel que soit le type de navigation, le facteur le plus important pour mesurer la qualité du service fourni par un système de streaming multimédia est le *temps de réponse* à une action de navigation. Ce temps correspond au laps de temps écoulé entre l'occurrence d'une action de navigation, par un usager, et l'instant où le système réagit à cette action du côté client en restituant la présentation à l'usager de manière conforme à son action. Plus ce temps de réponse est court, plus la restitution de la présentation est fluide et par conséquent la qualité de service fourni par le système s'accroît.

2.1.5.1 Navigation dépendante du document

Cette forme de navigation est conçue par l'auteur du document de façon à aider les lecteurs dans leur parcours du document. La navigation dépendante se matérialise *via* des *objets d'interaction*, appelés aussi *objets activables*, faisant partie de la spécification du document. Ainsi, une présentation multimédia contient des objets médias mais aussi des objets d'interaction.

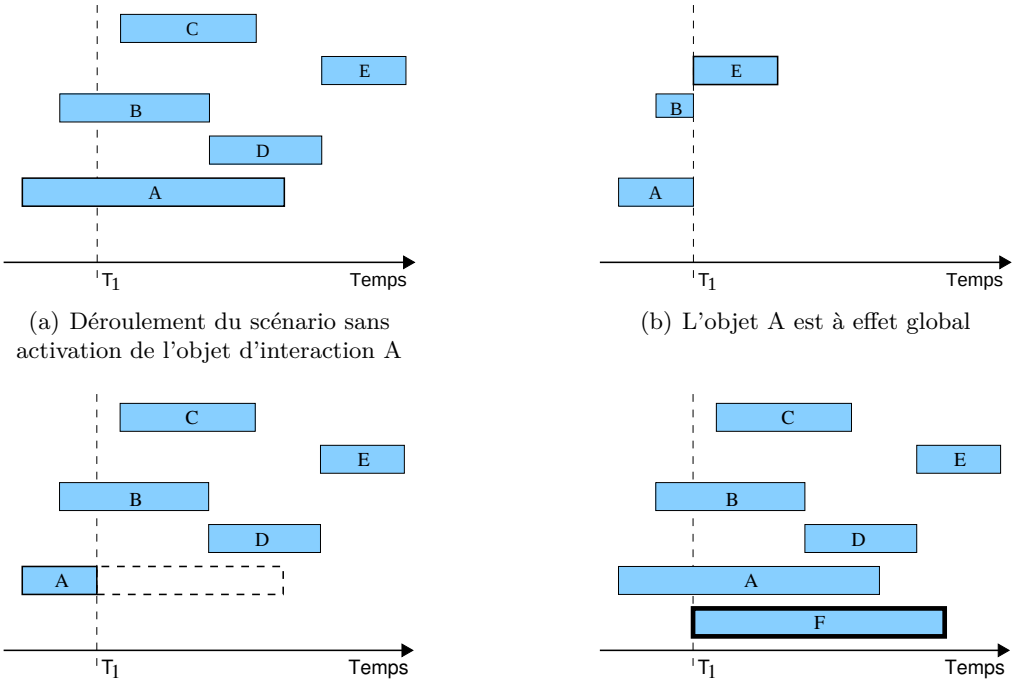
Dans notre exemple, (un cours de chimie), le bouton permet d'accéder directement aux notions présentées dans le cours sans passer par les préliminaires. Ce bouton apparaît au début des préliminaires et disparaît lors d'une activation par un usager permettant un accès direct à la première notion présentée dans le cours. En cas de non activation, le bouton reste affiché jusqu'à la fin des préliminaires, où il disparaît à l'instant T_4 (cf. Figure 2.3). Un autre exemple de tels objets d'interaction peut être un bouton permettant l'apparition, ou la disparition, d'un graphique illustrant la trajectoire d'un disque ou d'un javelot dans une présentation multimédia sur les jeux olympiques.

Les objets d'interaction tombent dans trois catégories en fonction de leur comportement, c'est-à-dire en fonction de la nature de l'effet qui leur est associé. La Figure 2.6 illustre ces cas.

- **Objets à effet global :** L'activation d'un objet à effet global (Figure 2.6.b) a pour conséquence de stopper la présentation de tous les objets en cours de restitution à cet instant pour démarrer la restitution des objets à partir d'un autre instant. C'est le cas du bouton dans le scénario représentant le cours de chimie (cf. Figure 2.3).

- **Objets à effet local :** L'activation d'un objet à effet local (Figure 2.6.c) a pour conséquence de ne stopper qu'une partie des objets en cours de restitution à un instant donné, mais n'agit pas sur le reste du scénario de telle sorte que les autres objets aient la possibilité de continuer leur exécution. Ce type d'interaction est plus difficile à gérer, car la synchronisation à gros grain est parfois difficile à assurer en présence d'objets d'interaction à effet local.

- **Objets à effet d'inclusion :** L'activation d'un objet à effet d'inclusion (Figure 2.6.d) a pour conséquence d'ajouter un sous-scénario à partir de l'instant courant de la présentation. Par exemple, un texte explicatif et un commentaire associés à une image affichée. Les objets du sous-scénario ne peuvent cependant pas être synchronisés avec les autres objets du document.



(a) Déroulement du scénario sans activation de l'objet d'interaction A

(b) L'objet A est à effet global

(c) L'objet A est à effet local

(d) L'objet A est à effet d'inclusion

Figure 2.6 – Les trois types d'interaction dépendante du document.

Afin de réduire le temps de réponse du système à des actions de navigation dépendante, les auteurs dans [FLSA+01] proposent de cacher les parties considérées « saillantes » par l'éditeur d'une présentation dans les proxies. Par exemple, les séquences contenant les buts dans une présentation d'un match de foot. Dans le cours de chimie qui nous sert d'exemple, ces parties correspondent aux notions présentées dans le cours. Lorsqu'un usager active le bouton dans le but de se passer des préliminaires, le temps de réponse à son action est ainsi réduit puisque la Notion #1 se trouve déjà dans le proxy. Il est à noter que cette technique est extrêmement gourmande en matière d'espace cache du côté du proxy.

2.1.5.2 Navigation indépendante du document

Cette forme de navigation est fournie *via* une barre de navigation permettant un accès aléatoire ou bien à travers des boutons de contrôle du temps, tels que les boutons de pause, de reprise ou d'accélération en avant ou en arrière. Ces fonctions de navigation sont indépendantes du document présenté ; elles n'apparaissent pas dans la spécification de son scénario.

Dans le cadre de la navigation indépendante, le parcours rapide d'une présentation, par le biais de fonctions de magnétoscope de type avance ou recul rapide, est celui qui pose le plus de problèmes lors de sa mise en œuvre. Pour illustrer ceci, nous prenons l'exemple d'un système de streaming de séquences vidéos et nous décrivons les techniques utilisées pour la mise en œuvre de différentes actions de navigation indépendantes des vidéos. Notons qu'une vidéo synchronisée à grain fin avec un commentaire audio constitue une forme très simplifiée de présentation multimédia, mais aussi la forme la plus courante.

Parcours accéléré

Nous distinguons trois techniques permettant d'effectuer un parcours rapide des séquences vidéos dans la littérature.

Lecture accélérée Une lecture accélérée des vidéos se traduit par une restitution à des cadences supérieures aux cadences nominales des vidéos. Le nombre d'images affichées par seconde est fonction du *coefficient d'accélération*. Supposons qu'une vidéo est cadencée à 15 images par seconde, la fréquence d'affichage lors d'une lecture accélérée est égale à $V \times 15$ images par secondes, où V représente le coefficient d'accélération (voir Figure 2.7). Cette technique est extrêmement gourmande en ressources à tous les niveaux : serveur, proxies, bande passante du réseau et des clients. Il est donc difficile d'envisager son emploi dans les systèmes de streaming d'aujourd'hui.

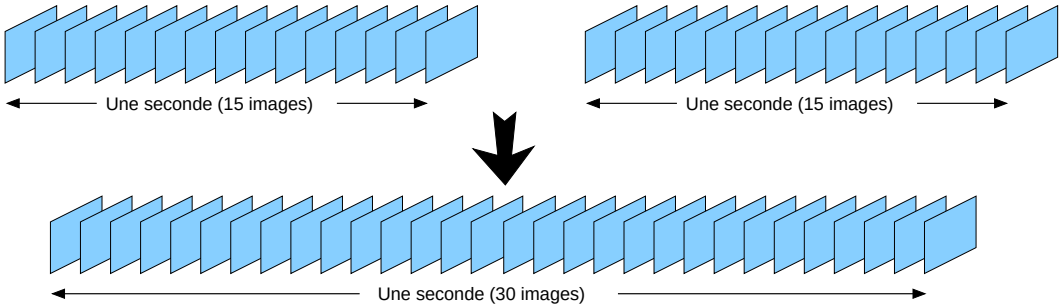


Figure 2.7 – Lecture accélérée, à coefficient 2, d'une vidéo cadencée à 15 images/seconde.

Accélération simulée Cette technique consiste à donner l'impression d'une restitution accélérée d'une vidéo en se privant d'afficher l'intégralité de ses images et d'afficher plutôt un sous-ensemble restreint de celles-ci. Les images choisies sont restituées à la cadence nominale de la vidéo. La sélection des images à préserver peut être effectuée de différentes manières. Elle peut être réalisée de façon aléatoire [TATH95], une image sur deux par exemple (voir Figure 2.8). Elle peut être basée sur le contenu des vidéos, que ce soit au niveau de changement des arrière-plans [ADHC94], d'objets saillants dans une séquence vidéo [KH00], ou bien d'activités dans

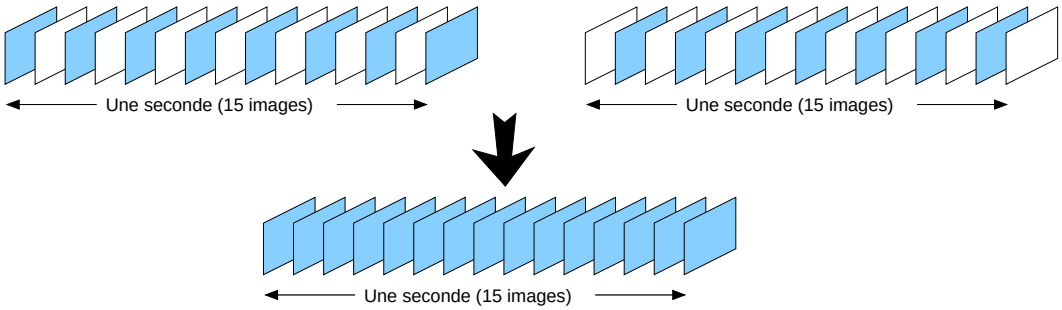


Figure 2.8 – Accélération simulée d'une vidéo cadencée à 15 images/seconde en sautant une image sur deux.

les images successives [VL98]. Ainsi, dans cette approche, lorsqu'un usager active une action d'avance rapide, le serveur lui transmet les images sélectionnées. Le client se charge ensuite de les afficher de manière successive.

Afin de faciliter la mise en œuvre de cette accélération simulée et notamment de réduire le temps de réponse à des interactions du type avance rapide et/ou recul rapide, Moser et al. proposent une stratégie de gestion du cache de serveur dénommée L/MRP [MKK95]. La politique L/MRP est une politique intégrée, c'est-à-dire qu'elle prend en charge les fonctionnalités de chargement et de remplacement des blocs de données dans la mémoire du serveur. L/MRP attribue *une valeur de pertinence* au remplacement ou au *préchargement* à chaque bloc de données dans une séquence vidéo. Les blocs ayant une grande valeur de pertinence sont préchargés dans la mémoire alors que les blocs ayant une faible valeur de pertinence sont sélectionnés pour être éventuellement remplacés. Pour définir ces valeurs de pertinence, L/MRP se base sur la définition des *ensembles d'interaction*, des ensembles de blocs de données à utiliser lors d'accès en mode avance rapide, recul, etc. Puis, elle introduit une fonction de calcul de la pertinence pour chaque ensemble. Lorsqu'un bloc appartient à plusieurs ensembles, elle conserve la valeur de pertinence la plus grande parmi les valeurs attribuées par les différentes fonctions.

La figure 2.9 illustre trois ensembles d'interactions et les valeurs de pertinences associées lors d'un accès en mode avance rapide avec un coefficient d'accélération égal à 2. L'ensemble *Passés* correspond aux blocs ayant déjà été envoyés au client et qui peuvent être potentiellement référencés dans le cas d'un retour en arrière. L'ensemble *Référencés* comprend les blocs qui seront restitués si l'utilisateur maintient son mode d'accès actuel. L'ensemble *Sautés* regroupe les blocs qui seront sautés au vu du mode d'accès actuel. Notons que la fonction de pertinence associée à l'ensemble *Passés* est linéaire tandis que les fonctions associées aux ensembles *Référencés* et *Sautés* sont du second degré. Il est à noter également que les blocs qui se trouvent proches du bloc en cours d'accès ont une pertinence plus élevée que ceux qui en sont éloignés.

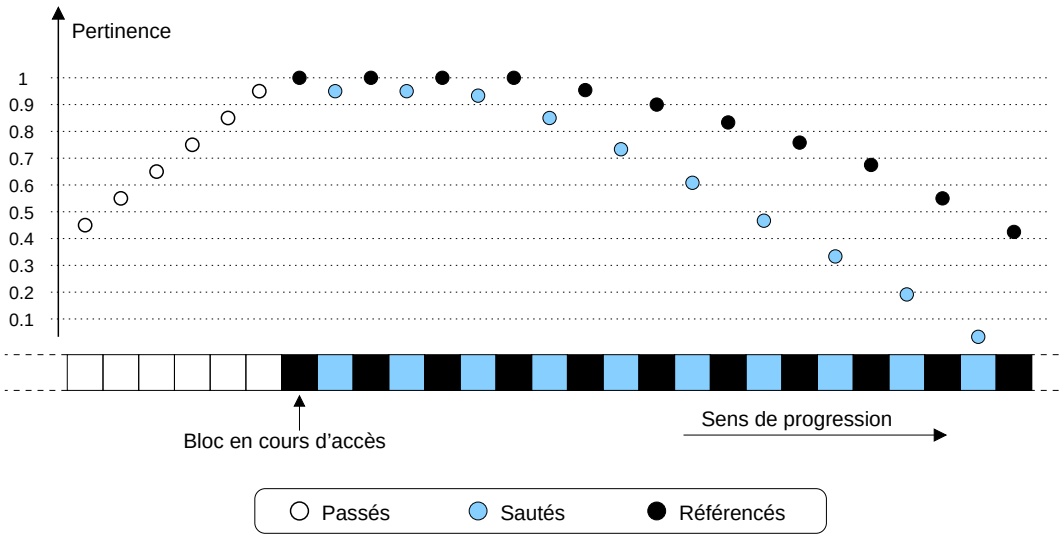


Figure 2.9 – L/MRP : exemple d'ensembles d'interaction avec les valeurs de pertinence associées aux blocs.

Utilisation de fichiers spéciaux Cette technique, plus récente, consiste à traiter les vidéos *a priori* afin de créer des fichiers les « condensant ». Il s'agit de fusionner les images contenant des activités, après avoir sauté les images qui n'en contiennent guère [RAPP06]. Cette technique est notamment utilisée pour résumer les vidéos issues de caméras de vidéo surveillance et de produire ainsi un *synopsis* de ces vidéos. En effet, ces vidéos sont caractérisées par des périodes d'inactivité extrêmement longues. De plus, les périodes d'activité sont ponctuelles et très éphémères. Un résumé ou un synopsis de ce genre de vidéos consiste donc à afficher, autant que possible, ces activités simultanément, même si elles ont eu lieu à des périodes différentes (voir Figure 2.10). Ainsi, une vidéo qui dure plusieurs heures est parcourue en quelques minutes.

Notons que cette technique peut aussi être utilisée pour résumer les évènements sportifs. Un match de football, par exemple, peut être résumé en quelques minutes en mettant en évidence les moments clefs du match. Les fichiers spéciaux résultants restent des vidéos qui peuvent être transmises et reçues comme des streams standards. Une action d'avance rapide d'une vidéo donnée entraînera le basculement du flux vers un fichier spécial associé à la vidéo jusqu'à la prochaine action d'avance normale. Toutefois, la transition entre une vidéo et sa version *résumée* n'est pas triviale.

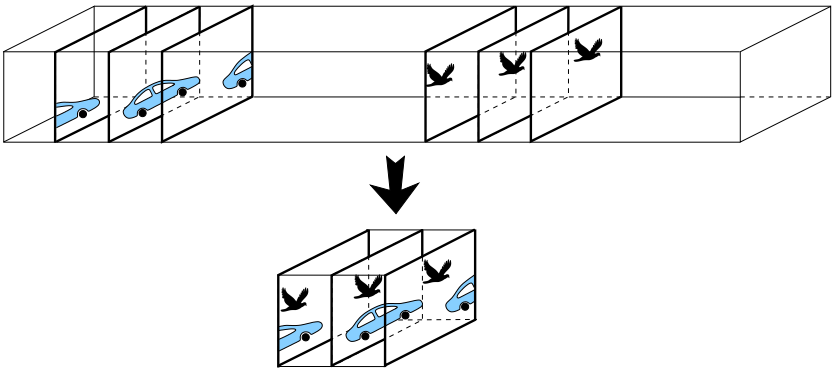


Figure 2.10 – Synopsis d'une vidéo.

2.2 Parcours rapide dans les présentations multimédias

Comme cela a été mentionné dans la section précédente, la navigation indépendante dans des présentations multimédias se limite aux fonctionnalités suivantes : arrêt, mise en pause et reprise. Le parcours rapide de ces présentations est absent des systèmes de streaming actuels. En effet, sa mise en œuvre soulève de nombreux problèmes conceptuels et techniques. Toutefois, la navigation accélérée dans des présentations multimédias est fortement souhaitée et même nécessaire. À titre d'exemple, un étudiant suivant le cours de chimie à distance, présenté dans la Section 2.1.1, souhaiterait avancer rapidement aux principales notions expliquées dans le cours. Un autre pourrait vouloir reculer pour revoir les préliminaires. Afin de fournir aux étudiants une utilisation ergonomique du système, il est indispensable que ce dernier leur fournisse des fonctionnalités permettant d'avancer et/ou reculer rapidement dans le cours.

Cette section commence par décrire les problèmes qui empêchent la mise en pratique de la navigation accélérée dans des présentations multimédias. Elle se poursuit par une description d'une nouvelle sémantique que nous proposons pour surmonter les limitations d'une navigation accélérée « conventionnelle ». L'atout de la sémantique proposée est qu'elle conserve la compréhensibilité d'une présentation lors d'un parcours rapide de celle-ci. Puis nous présentons la mise en œuvre de notre approche par l'utilisation d'un cache de lecture. Nous détaillons une stratégie, que nous avons développée et baptisée CPS [1], pour la gestion de ce cache. L'objectif premier de notre politique de gestion est de garantir une bonne qualité de service en terme de temps de réponse à une action de type avance rapide ou recul rapide.

2.2.1 Problématique

Il est inapproprié de vouloir réaliser la navigation accélérée dans une présentation multimédia en utilisant la même méthode que celle employée pour accélérer les vidéos. En effet, une telle réalisation de la navigation accélérée implique de nombreux désavantages :

1. Elle implique une consommation de ressources extrêmement élevée en termes de bande passante réseau, de puissance de calcul et de taille de mémoire au niveau du serveur, des proxies, du client, etc.
2. Elle provoque une restitution inintelligible de la présentation qui perd alors son sens. À titre d'exemple, un son sera incompréhensible, un texte sera présenté pendant une période insuffisante pour le lire, etc.
3. Elle entraîne le risque qu'un usager ne s'aperçoive pas de la présence d'objets importants d'interaction et notamment ceux à effet d'inclusion. Le scénario rédigé par l'auteur de la présentation ne sera donc pas pris en compte.

D'ailleurs, l'objectif que vise un usager lorsqu'il active une action d'avance (ou recul) rapide est d'accéder rapidement à une information souhaitée. Dans ce sens, les interactions de type avance rapide ou recul rapide sont liées au contenu d'une présentation.

Au vu de cette analyse, nous concluons que la sémantique même de la navigation accélérée dans une présentation multimédia doit être reconsidérée. Le parcours rapide d'une présentation doit prendre en compte son contenu en matière de concepts qui y sont abordés. En particulier, la

1. Content-Based Prefetching Strategy.

compréhension de ce contenu doit être conservée lors d'une navigation accélérée. En même temps, un accès rapide aux différentes informations doit être possible lors de la navigation accélérée dans une présentation multimédia. Dans la suite, nous présentons une nouvelle sémantique du parcours rapide de présentations multimédias, qui relie les fonctionnalités d'avance rapide et de recul rapide au contenu de ces présentations.

2.2.2 Parcours rapide proposé

Une présentation multimédia, comme tout document, est une suite cohérente d'informations ou d'*idées* qui a pour but de proposer ou de défendre un concept donné, de raconter une histoire quelconque, etc. Lors d'une navigation accélérée dans une présentation, plutôt que d'accélérer la restitution de différents objets et de courir le risque qu'ils ne deviennent incompréhensibles, nous proposons d'effectuer des sauts d'une idée vers une autre, vers l'avant ou vers l'arrière. Ainsi, la navigation accélérée que nous proposons diffère fondamentalement de l'accélération « conventionnelle » connue pour les séquences vidéos (cf. Section 2.1.5.2). Elle se traduit par des sauts dans le temps, plutôt que par une accélération.

La concrétisation de ce nouveau parcours rapide passe par la réponse aux deux questions suivantes :

- comment les idées d'une présentation sont-elles détectées ?
- comment ce parcours rapide est-il mis en pratique ?

2.2.2.1 Détection des idées dans une présentation

Les idées au sein d'une présentation multimédia peuvent être réparties de deux façons, qui peuvent être employées conjointement l'une avec l'autre : par l'auteur de la présentation durant l'étape de rédaction/composition, mais aussi de manière automatique à partir de méta-données associées à la présentation, comme le scénario. Dans la suite, nous décrivons comment nous dépistons les idées au sein d'une présentation, en tirant profit de la multiplicité des objets qui s'y trouvent.

Afin de constituer un panel d'idées d'une présentation multimédia nous partons des observation suivantes :

- Le début de chaque objet, média ou d'interaction, annonce généralement le début d'une idée. Dans l'exemple illustré par la Figure 2.11, l'apparition de l'image indique le début de l'idée #2 ; le démarrage de la vidéo signale le début d'une autre, l'idée #3. L'apparition du bouton, qui est un objet d'interaction, déclare elle aussi le début d'une idée, l'idée #1.

- La fin d'un objet média annonce la fin d'une idée et par conséquent déclare le début d'une autre, sauf dans le cas où l'idée terminée se trouve être la dernière idée dans une présentation. Par exemple, la fin de la vidéo, dans la Figure 2.11, déclare la fin de la dernière idée détectée, en l'occurrence l'idée #4, et indique donc le début de l'idée #5.

Notons que les données de description et d'indexation associées aux médias contenus peuvent également être utilisées afin de dériver les idées intrinsèques à ces médias. Ces données, considérées elles aussi comme méta-données, sont rendues disponibles grâce à des normes émergentes telles que le standard MPEG-7 [Mar02, MKP02].

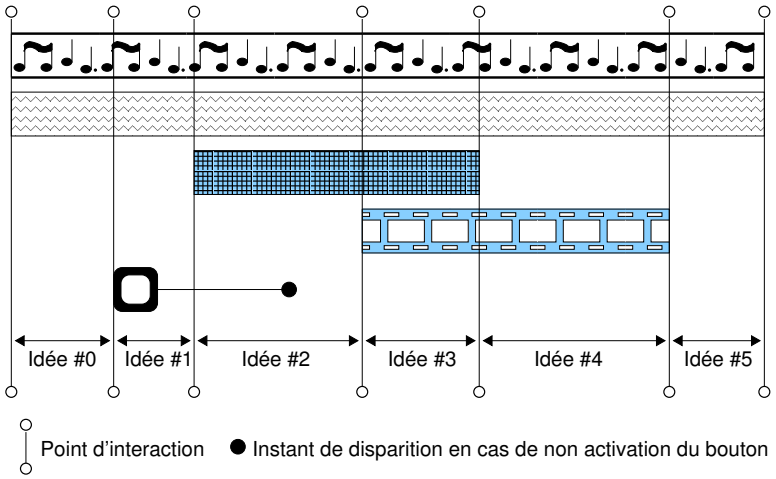


Figure 2.11 – Exemple d'idées dans une présentation multimédia.

Afin de délimiter les idées au sein d'une présentation, nous marquons le début de chaque idée par un *point d'interaction* (voir Figure 2.11). Ces points sont des marques temporelles que nous utiliserons pour effectuer les interactions de parcours rapide. Ils sont positionnés de manière automatique [2] sur l'axe du temps comme suit : un point d'interaction est placé à chaque

- point d'indexation déduit des données de description des médias contenus ;
- début d'objet média ou d'interaction ;
- fin d'objet média.

Comme nous allons voir ultérieurement, le fait de marquer le début d'objets d'interaction, en tant que point d'interaction, garantit aux usagers la possibilité d'activer ou non ces objets. En effet, la sémantique que nous proposons assure cette possibilité, puisqu'elle n'effectue pas d'accélération empêchant un usager d'activer un objet d'interaction. D'un autre côté, la fin d'un objet d'interaction n'est pas marquée avec un point d'interaction, car elle n'est pas connue à l'avance. D'ailleurs, cette fin est conditionnée par l'activation de l'objet par un usager.

Il est clair que la manière décrite ci-dessus pour dériver les idées incluses dans une présentation permet la détection de toute idée introduite par le début ou la fin d'un objet appartenant à la présentation. Les données d'indexation permettent de leur côté de détecter les idées enfouies à l'intérieur des médias continus.

Définition. Soit PI l'ensemble des points d'interaction placés sur l'axe du temps d'une présentation multimédia (PM) de la manière décrite ci-dessus.

$$PI = \{PI_0,\ PI_1,\ \cdots,\ PI_n\} \quad ; \quad PI_j < PI_{j+1} \quad \forall j \in \{0,\ 1,\ \cdots,\ n-1\} \tag{2.1}$$

Ces points d'interaction sont ordonnés temporellement et délimitent n idées dans la présentation PM. Notons que ces idées n'ont pas forcément des durées identiques. La présentation PM est dorénavant vue comme une suite ordonnée d'idées et nous la dénotons de la façon suivante :

$$PM = \{I_0,\ I_1,\ \cdots,\ I_{n-1}\} \tag{2.2}$$

où l'idée I_j est délimitée par les deux points d'interaction : P_j et P_{j+1}, $\forall j \in \{0,\ 1,\ \cdots,\ n-1\}$.

2. En complément à ces points d'interaction, l'auteur de la présentation peut en ajouter d'autres, durant l'étape de rédaction/composition.

2.2.2.2 Gestion du nouveau parcours rapide

La mise en pratique de la nouvelle sémantique du parcours rapide s'appuie sur les points d'interaction utilisés pour marquer les débuts d'idées dans une présentation. Un saut vers une idée se traduit par la restitution de la présentation à partir du début de cette idée, c'est-à-dire à partir du point d'interaction marquant son début (voir Figure 2.12). Ce principe nous permet de définir quatre fonctionnalités de navigation accélérée dans une présentation : redémarrage de la présentation, reprise d'une idée, avance rapide vers l'idée suivante ou bien recul rapide vers l'idée précédente.

Supposons que l'on ait détecté n idées dans une présentation PM en cours de restitution à partir de la position T dans son axe du temps : $T \in [\ PI_0\ ,\ PI_n\ [$, il existe alors une idée I_j en cours de restitution, c'est-à-dire qu'il existe $j \in \{0,\ 1,\ \cdots,\ n-1\}$ tel que $T \in [\ PI_j\ ,\ PI_{j+1}\ [$. Les différentes fonctionnalités permettant la navigation accélérée dans la présentation PM sont ainsi mises en pratique par le biais de points d'interaction comme suit :

- **Redémarrage de la présentation** : si une telle interaction est activée par un usager, un saut vers le début de la présentation est effectué. Ce saut correspond à un saut vers le début de la première idée de la présentation, c'est-à-dire un saut vers le point d'interaction PI_0.

- **Reprise d'une idée** : si une telle interaction est activée par un usager, un saut vers le début de l'idée en cours de restitution, I_j, est effectué. Ce saut correspond à un saut vers le point d'interaction PI_j.

- **Avance rapide** : si une telle interaction est activée par un usager, un saut vers le début de l'idée suivante, I_{j+1}, est effectué. Ce saut correspond à un saut vers le point d'interaction PI_{j+1}. Notons que cette interaction se traduit par l'arrêt de la présentation dans le cas où l'idée en cours de restitution se trouve être la dernière $(j = n-1)$; il n'y a pas d'idée suivante vers laquelle on peut effectuer un saut.

- **Recul rapide** : si une telle interaction est activée par un usager, un saut vers le début de l'idée précédente, I_{j-1}, est effectué. Ce saut correspond à un saut vers le point d'interaction PI_{j-1}. À l'instar de l'avance rapide, cette interaction se traduit par l'arrêt de la présentation dans le cas où l'idée en cours de restitution se trouve être la première $(j = 0)$; il n'y a pas d'idée précédente vers laquelle on peut revenir.

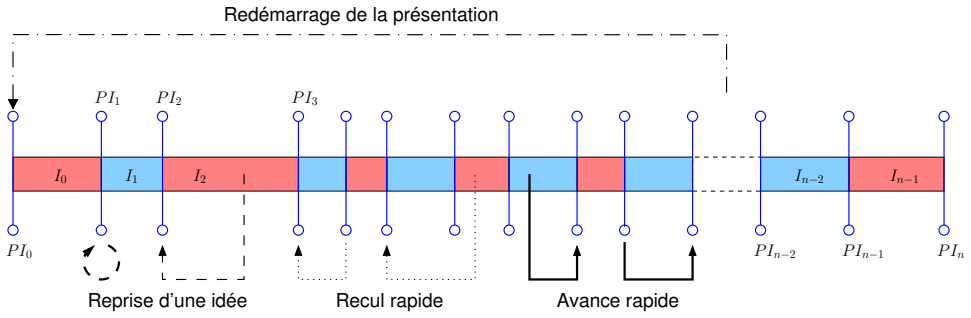


Figure 2.12 – Parcours rapide d'une présentation multimédia au travers des points d'interaction.

2.2.3 Mise en œuvre

Comme nous venons de le voir, le parcours rapide proposé se traduit par des sauts dans le temps. La destination de ces sauts, qui diffère en fonction de l'interaction activée, est toujours définie par un point d'interaction. La mise en œuvre de ce parcours est donc simple ; il suffit que le lecteur de présentation, et plus particulièrement le contrôleur, restitue la présentation à partir du point d'interaction correspondant à l'interaction activée. Bien entendu, la restitution depuis la nouvelle position obéit toujours aux contraintes de synchronisation intra et inter objets spécifiées par le scénario de la présentation. Toutefois, la fluidité de la restitution de celle-ci, lors d'une interaction de parcours rapide, est pénalisée par le délai d'acheminement des données à restituer. Ces données, issues de plusieurs objets, doivent en effet être acheminées à travers le réseau vers le cache de lecture du client afin qu'elles soient ensuite décodées puis restituées. Ce délai d'acheminement augmente le temps de latence du système à de telles interactions et détériore donc la qualité du service offert par ce dernier.

Afin de réduire le temps de latence du système aux interactions de parcours rapide proposées, nous optons pour l'élaboration d'une *politique de préchargement* adaptée à la nouvelle sémantique du parcours rapide. Le préchargement, appelé aussi *prefetching*, est une technique bien connue dans les systèmes de streaming. Il s'agit de prédire les données auxquelles on aura accès ultérieurement et de les charger au préalable dans des caches au sein du système de streaming, avant d'en avoir effectivement besoin. Le préchargement, lorsqu'il est pertinent, permet d'améliorer sensiblement les performances des systèmes de streaming. Ce fait est valable quel que soit le type de cache dans lequel on précharge les données.

En effet, on trouve des techniques de préchargement à tous les niveaux d'un système de streaming : serveur, proxy et client. D'abord, les techniques de préchargement au niveau du serveur visent la réduction des accès à son système de stockage. À titre d'exemple, nous citons L/MRP qui intègre l'accélération simulée de séquences audiovisuelles (cf. Section 2.1.5.2) ou alors la politique décrite dans [MKB02] qui est spécialement développée pour les serveurs parallèles de séquences vidéo. Ensuite, au niveau des proxies, les techniques d'insertion et de remplacement de contenus ou de parties de contenus, et les techniques de mise en cache de leurs préfixes (cf. Section 1.4.3.1) sont toutes considérées comme des techniques de préchargement. Ce sont des approches prédictives dont l'objectif est que les données soient disponibles dans les proxies au moment où on en aura besoin. Ainsi, le trafic entre le serveur et les proxies se trouve réduit et le délai d'acheminement des données aux clients diminué. Enfin, les techniques de gestion de cache de lecture, décrites dans la Section 1.4.1, ne sont rien d'autre que des techniques de préchargement dont le but est de combattre l'effet de la gigue dans les réseaux à commutation de paquets et donc d'assurer la fluidité de la restitution des contenus audiovisuels.

La politique de gestion de cache que nous présentons dans la section suivante est spécialement conçue pour élargir les fonctionnalités du cache de lecture et permettre au gestionnaire de ce cache (voir Figure 2.5) de prendre en charge le parcours rapide décrit précédemment. Puisque ce parcours est étroitement lié au contenu d'une présentation multimédia, nous avons baptisé cette politique de gestion de cache CPS, acronyme de « Content-based Prefetching Strategy ».

Notons que CPS est une politique intégrée au sens où elle intègre les fonctionnalités de chargement/préchargement et de remplacement de blocs de données. Elle permet d'améliorer nettement la qualité de service offert par le système ; elle réduit considérablement le temps de réponse aux interactions du parcours rapide proposé. Les résultats expérimentaux, présentés dans la Section 2.3, le confirment.

2.2.4 Politique de préchargement : CPS

La politique CPS est une politique de gestion de cache adaptée spécialement au parcours rapide des présentations multimédias, par des sauts dans le temps vers le début d'une idée de la présentation. Elle s'assure de la présence des blocs de données dont on aura probablement besoin ultérieurement, que ce besoin soit issu d'un accès « normal » ou suite à une interaction de parcours rapide. Pour cela, elle commence par diviser la présentation en *unités de présentation*. Ensuite, à l'aide de *fonctions de pertinence*, elle attribue à chaque unité une valeur reflétant sa pertinence au remplacement ou au préchargement. CPS agit en conséquence et précharge les blocs auxquels l'accès est probable dans le cache de lecture à la place des blocs auxquels l'accès est moins probable. Les détails de cette politique sont décrits dans les paragraphes suivants.

2.2.4.1 Unité de présentation

Une présentation multimédia est décomposée en unités de présentation d'une même durée égale à un *laps de temps*, qui est un paramètre de configuration du lecteur de présentations. Une unité de présentation comprend des parties d'objets à restituer dans un laps de temps (voir Figure 2.13). La taille d'une unité de présentation est donc variable ; elle est tributaire de la taille des données, issues de plusieurs objets, incluses dans l'unité. En outre, le nombre d'unités de présentation dans une idée est variable lui aussi ; il est fonction de la durée de l'idée en question.

Les unités de présentation (UP) sont identifiées par leur position relative au début de l'idée dont elles font partie. Soit l_k le nombre d'unités incluses dans une idée I_k d'une présentation PM contenant n idées, $k \in \{0, 1, \cdots, n-1\}$. Ainsi, toute idée I_k peut être vue comme une suite d'unités de présentation :

$$I_k = \left\{ UP_0^k,\ UP_1^k,\ \cdots,\ UP_{l_k-1}^k \right\} , \quad k \in \{0, 1, \cdots, n-1\} \tag{2.3}$$

La présentation multimédia peut elle aussi être exprimée en tant qu'une suite d'unités de présentation comme suit :

$$PM = \left\{ UP_0^0,\ \cdots,\ UP_{l_0}^0,\ UP_0^1,\ \cdots,\ UP_{l_1-1}^1,\ \cdots,\ UP_0^{n-1},\ \cdots,\ UP_{l_{n-1}-1}^{n-1} \right\} \tag{2.4}$$

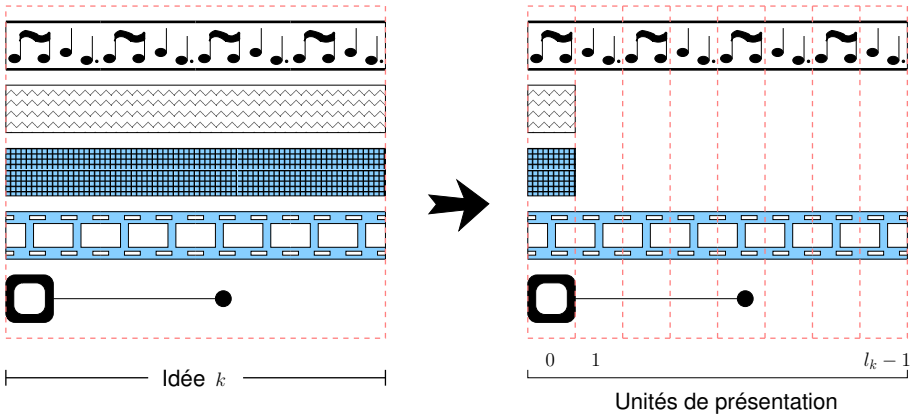


Figure 2.13 – Décomposition d'une idée en unités de présentation.

Notons que les objets d'interaction faisant partie d'une idée I_k ne sont pas divisés lors de la décomposition en unités de présentation. Ces objets sont inclus dans la première unité de présentation, UP_0^k, de l'idée, I_k, dont ils font partie. Lorsqu'un objet d'interaction est affiché, au début de l'idée, il le restera jusqu'à son activation par un usager ou alors jusqu'à l'instant de sa disparition défini par l'auteur de la présentation. Il en va de même pour les médias statiques de type image par exemple, ils sont inclus dans la première unité de l'idée dont ils font partie. Une fois affichés ils le resteront jusqu'à la fin de l'idée en question. Les médias textuels sous forme de flux, une transcription d'un discours audio par exemple, ont un volume négligeable par rapport aux flux audiovisuels. À l'instar d'une image ou d'un texte statique, les flux textuels sont donc inclus dans la première unité de l'idée dont ils font partie ; ils resteront dans le cache tant que l'idée est en cours de restitution. Ainsi, les autres unités de présentation : $\{UP_1^k,\ UP_2^k,\ \cdots,\ UP_{l_k-1}^k\}$, de l'idée I_k, contiennent exclusivement des données audiovisuelles à restituer au fur et à mesure (voir Figure 2.13).

2.2.4.2 Fonction de pertinence

CPS s'appuie sur des valeurs associées à chaque unité de présentation pour évaluer sa pertinence au préchargement ou au remplacement. Ces valeurs sont attribuées aux différentes unités de présentation à l'aide des fonctions de pertinence décrites ci-dessous.

Les fonctions de pertinence que nous utilisons sont décroissantes et monotones. Les valeurs les plus grandes sont attribuées à l'unité de présentation en cours de restitution ainsi qu'aux premières unités de chaque idée. Ceci est motivé par le fait qu'un utilisateur peut, à tout moment, activer l'une des fonctionnalités du parcours rapide proposé. Si une telle interaction est activée, l'idée correspondante devra être restituée au plus vite, voire immédiatement. On aura alors besoin de la première unité de cette idée aussitôt ; l'unité en question devra donc être disponible dans le cache. Quant aux unités qui suivent les premières unités de chaque idée, leur valeur de pertinence décroît avec l'éloignement de la première unité de l'idée dont elles font partie (voir Figure 2.15).

Soit UP_T^h l'unité de présentation en cours de restitution au sein de l'idée I_h dans une présentation PM contenant n idées. Le préchargement de l'unité UP_{T+j}^h dans le cache est plus pertinent que le préchargement de l'unité UP_{T+j+1}^h, $j \in \{ T+1,\ T+2,\ \cdots,\ l_h-1\}$. Il en va de même pour les unités de présentation appartenant à une idée I_k de la présentation, $k \in \{ 0,\ 1,\ \cdots,\ n-1\}$. Le préchargement de l'unité UP_j^k dans le cache est plus pertinent que le préchargement de l'unité UP_{j+1}^k, $j \in \{ 0,\ 1,\ \cdots,\ l_k-1\}$. Ceci est du à l'accès séquentiel aux unités de présentation au sein d'une idée, sauf en cas d'interaction de la part d'un utilisateur.

Afin de garantir des valeurs de pertinence nettement élevées aux unités « proches » de l'unité en cours de restitution ou « proches » de la première unité d'une idée par rapport aux unités qui en sont « éloignées », la décroissance des fonctions de pertinence doit être relativement lente. Ainsi, des fonctions linéairement décroissantes ne sont pas appropriées dans ce cadre. Nous avons alors opté pour des fonctions du deuxième degré, qui ont un temps d'exécution polynomial inférieur à celui d'une fonction de troisième degré, chose qui est non négligeable puisque les fonctions de pertinence seront fréquemment utilisées. La parabole de base que nous utilisons est représentée par l'équation :

$$x^2 = -4\ (y-1) \qquad \text{ou d'une façon équivalente} \qquad y = \frac{4-x^2}{4} \tag{2.5}$$

Plus précisément, nous utilisons une partie descendante de cette parabole ($x \in [0\ ;\ 2]$), cette partie est tracée par la ligne pointillée dans la Figure 2.14. Comme on peut voir sur la figure, la forme parabolique de la partie en question correspond tout à fait à nos besoins précisés précédemment. Néanmoins, nous l'avons modifié afin d'obtenir une fonction dont les valeurs de pertinence sont positives pour toutes les unités de présentation, tout en gardant la même forme parabolique.

Soit L le maximum des cardinaux des idées de la présentation PM, en considérant une idée comme un ensemble d'unités (cf. Relation 2.3), c'est-à-dire le nombre maximal d'unités inclues dans une idée de la présentation :

$$L = \max\ \{\ l_0,\ l_1,\ \cdots\ ,\ l_{n-1}\ \} \quad ; \quad L > 2 \tag{2.6}$$

La fonction de base, que nous utilisons pour calculer la pertinence des différentes unités de présentations appartenant à une idée, est alors obtenue à partir de la parabole (2.5) avec une homothétie de rapport $L/2$ (voir Figure 2.14) :

$$x^2 = -2 \cdot L\ (y - \frac{2}{L}) \qquad \text{ou d'une façon équivalente} \qquad y = \frac{L}{2} - \frac{x^2}{2 \cdot L} \tag{2.7}$$

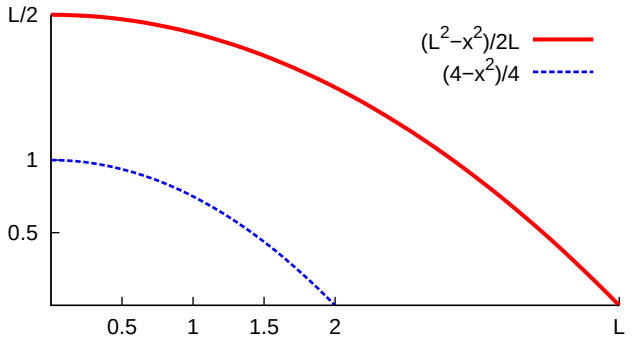


Figure 2.14 – Paraboles de base, utilisées dans les fonctions de pertinence.

Les fonctions de pertinence adoptées par la politique CPS sont ainsi obtenues à partir de la parabole (2.7), qui est conçue pour répondre aux besoins spécifiques du parcours rapide d'une présentation multimédia. Soit A l'ensemble des unités de présentation qui suivent l'unité en cours de restitution UP_T^h au sein de la même idée I_h, y compris l'unité en cours de restitution :

$$A = \left\{UP_T^h,\ UP_{T+1}^h,\ \cdots\ ,\ UP_{l_h}^h\right\} \tag{2.8}$$

La définition formelle de la fonction de pertinence utilisée pour attribuer des valeurs de pertinence aux unités appartenant à l'ensemble A est alors donnée par (voir Figure 2.15) :

$$\textbf{\textit{Pertinence}}\ (x) : \left| \begin{array}{ccl} A & \longrightarrow & \mathbb{R}^{+\star} \\ UP_x^h & \longmapsto & \textit{Pertinence}\ (UP_x^h) = \frac{L}{2} - \frac{(x-T)^2}{2 \cdot L} \end{array} \right. \tag{2.9}$$

$$x\ \in\ \{T,\ T+1,\ \cdots,\ l_h - 1\}$$

Quant aux autres unités de la présentation, pour toute unité sauf celle appartenant à l'ensemble A, la définition formelle des fonctions de pertinence est donnée par (voir Figure 2.15) :

$$\textbf{\textit{Pertinence}}\ (k,x) : \left| \begin{array}{ccc} PM - A & \longrightarrow & \mathbb{R}^{+\star} \\ UP_x^k & \longmapsto & Pertinence\left(UP_x^k\right) = \frac{L}{2} - \frac{x^2}{2 \cdot L} \end{array} \right. \tag{2.10}$$

$$k \in \{0,\ 1,\ \cdots,\ n-1\}, \quad x \in \{0,\ 1,\ \cdots,\ l_k - 1\}$$

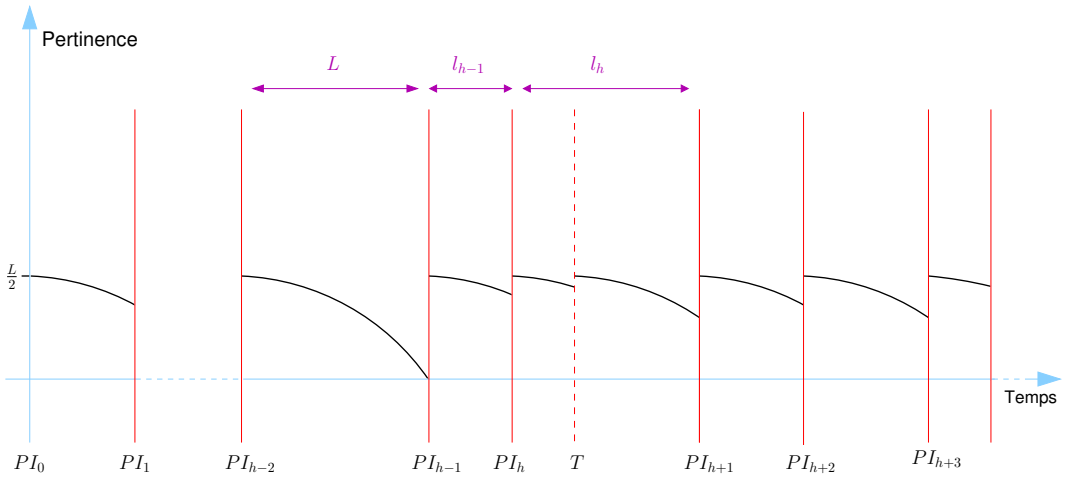


Figure 2.15 – Fonctions de pertinence dans la politique CPS.

2.2.4.3 Gestion de cache

La politique CPS attribue des valeurs de pertinence aux différentes unités d'une présentation multimédia en utilisant les fonctions (2.9) et (2.10). Elle se sert ensuite de ces valeurs pour décider quelles unités seront candidates au préchargement. Lorsque le cache est plein, elle utilise ces même valeurs pour désigner les unités qui seront remplacées. Plus la valeur de pertinence attribuée à une unité est faible, plus sa priorité à être préchargée est diminuée et plus la probabilité qu'elle soit remplacée augmente si elle se trouvait en cache au moment où on a besoin d'espace mémoire.

CPS gère le cache de manière à ce que les unités auxquelles l'accès est fort probable soient préchargées dans le cache en premier. Afin d'accomplir cela, elle introduit la notion de *priorité* entre les unités à précharger. Cette priorité est incarnée par le paramètre α qui indique le *niveau d'interactivité* pris en compte par CPS. En supposant que l'unité UP_T^h est en cours de restitution au sein de l'idée I_h dans une présentation PM contenant n idées, les différents niveaux d'interactivité correspondent aux valeurs suivantes de α :

- $\boldsymbol{\alpha = 0}$, indique que CPS ne prend pas en compte les fonctionnalités de parcours rapide. Elle précharge uniquement les unités à restituer dans le futur en cas de lecture « normale », c'est-à-dire les unités dont les valeurs de pertinence sont les plus élevées et qui suivent l'unité en cours de restitution :

$$\left\{UP_T^h,\ UP_{T+1}^h,\ \cdots\right\}$$

- $\boldsymbol{\alpha = 1}$ indique que CPS commence à prendre en compte les fonctionnalités du parcours rapide. En plus des unités qui correspondent à $\alpha = 0$, CPS précharge les unités dont les valeurs de pertinence sont les plus élevées et qui succèdent aux premières unités de la première idée de la présentation, de l'idée en cours de restitution, de l'idée suivante et de l'idée précédente (voir Figure 2.16). De manière formelle, CPS précharge les unités suivantes :

$$\begin{aligned}\left\{UP_T^h,\ UP_{T+1}^h,\ \cdots\right\} \bigcup\ & \left\{UP_0^0,\ UP_1^0,\ \cdots\right\} \bigcup \left\{UP_0^h,\ UP_1^h,\ \cdots\right\} \\ \bigcup\ & \left\{UP_0^{h+1},\ UP_1^{h+1},\ \cdots\right\} \bigcup \left\{UP_0^{h-1},\ UP_1^{h-1},\ \cdots\right\}\end{aligned}$$

- $\boldsymbol{\alpha = 2}$ indique aussi que CPS prend en compte les fonctionnalités du parcours rapide. En plus des unités dont la valeur de pertinence est la plus élevée et qui correspondent aux valeurs du paramètre $\alpha = 1$, CPS précharge les unités dont la valeur de pertinence est la plus élevée et qui succèdent aux premières unités d'une idée qui suit et d'une idée qui précède (voir Figure 2.16). De manière formelle, CPS précharge les unités suivantes :

$$\begin{aligned}\left\{UP_T^h,\ UP_{T+1}^h,\ \cdots\right\} \bigcup\ & \left\{UP_0^0,\ UP_1^0,\ \cdots\right\} \bigcup \left\{UP_0^h,\ UP_1^h,\ \cdots\right\} \\ \bigcup\ & \left\{UP_0^{h+1},\ UP_1^{h+1},\ \cdots\right\} \bigcup \left\{UP_0^{h-1},\ UP_1^{h-1},\ \cdots\right\} \\ \bigcup\ & \left\{UP_0^{h+2},\ UP_1^{h+2},\ \cdots\right\} \bigcup \left\{UP_0^{h-2},\ UP_1^{h-2},\ \cdots\right\}\end{aligned}$$

- Et ainsi de suite, tant qu'il y a des idées dans la présentation vers lesquelles l'usager peut effectuer des sauts.

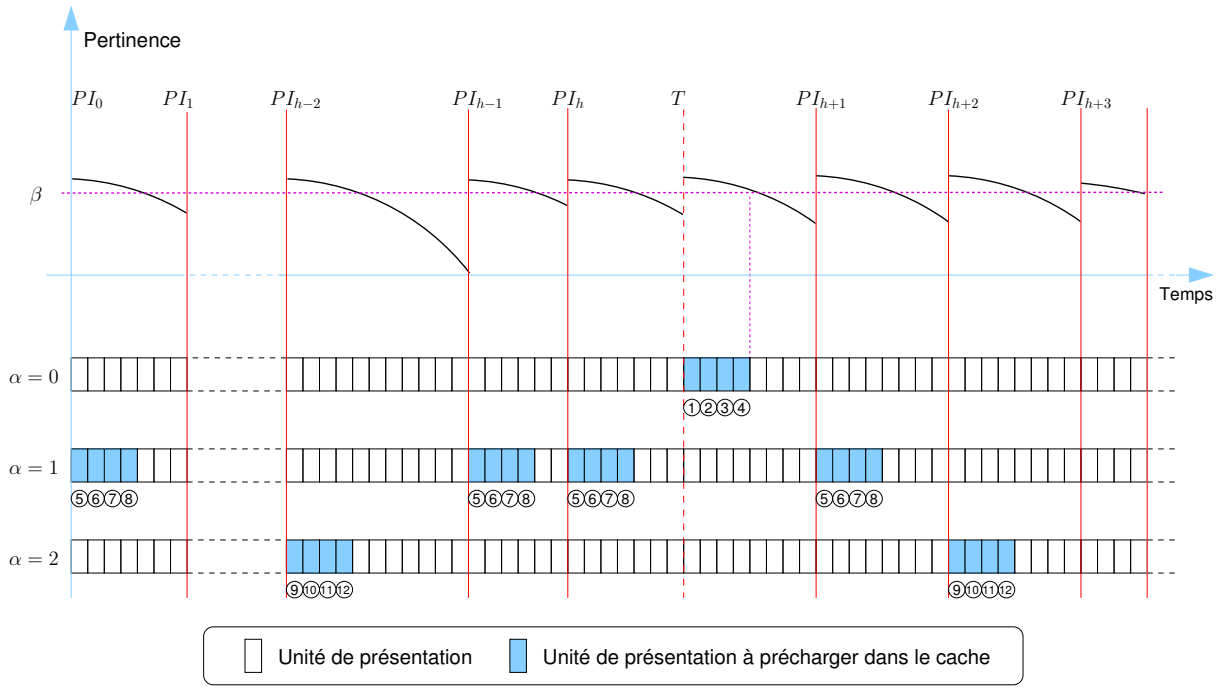


Figure 2.16 – CPS : exemple d'unités à préchargées avec une valeur donnée de β et plusieurs valeurs de α.

Le paramètre α permet de configurer CPS pour différentes applications de streaming, exigeant divers niveaux d'interactivité. Par exemple, les utilisateurs dans des applications de boutiques virtuelles sont beaucoup plus interactifs que des étudiants dans une application d'enseignement à distance. Ils font plus d'*aller-retour* entre les idées d'une présentation multimédia. Il est donc judicieux de donner des valeurs élevées au paramètre α dans les applications fortement interactives, et des valeurs moins élevées dans les applications moins interactives.

CPS précharge les unités dans le cache en commençant par les unités faisant partie des niveaux d'interactivité inférieurs. Elle précharge les unités qui correspondent au niveau d'interactivité $\alpha = 0$ en premier. Ensuite, elle précharge les unités qui correspondent aux niveaux supérieurs $\alpha = 1$, puis $\alpha = 2$, et ainsi de suite.

Le préchargement des unités de chaque niveau commence toujours par les unités dont la valeur de pertinence est la plus élevée, puis par les unités ayant des valeurs de pertinence moins élevées. CPS précharge uniquement les unités dont la valeur de pertinence dépasse un certain

Algorithme 1 – Politique de gestion de cache : CPS

```
   Paramètres : α_max, β
   ▷ α_temp : valeur temporaire du paramètre α
   ▷ F : fragment de données
   ▷ F_remplacer : fragment de données à remplacer
   ▷ Date : valeur pour marquer la date de chargement d'un fragment dans le cache
 1 Date ← temps courant
 2 α_temp ← 0
 3 tant que α_temp ≤ α_max faire
 4 |   pour tous les UP_i dont α = α_temp faire
 5 |   |   calculer Pertinence (UP_i) en utilisant les fonctions (2.9) et (2.10)
 6 |   |   Pertinence ← Pertinence (UP_i)
 7 |   |   pour tous les UP_i dont Pertinence > β faire
 8 |   |   |   pour tous les fragments F de UP_i faire
 9 |   |   |   |   associer Pertinence à F
10 |   |   |   |   associer Date à F
11 |   |   |   |   F_remplacer ← trouver un fragment à remplacer
   |   |   |   |   ▷ F_remplacer : celui dont la valeur de pertinence est la plus petite et dont la
   |   |   |   |     date de chargement dans le cache est à la fois la plus ancienne et
   |   |   |   |     inférieure à Date ; NULL si un tel fragment n'existe pas dans le cache
12 |   |   |   |   si F_remplacer ≠ NULL alors
13 |   |   |   |   |   remplacer F_remplacer avec F
14 |   |   |   |   sinon
15 |   |   |   |   |   sortie de l'algorithme
16 |   |   |   |   fin
17 |   |   |   fin
18 |   |   fin
19 |   fin
20 |   α_temp ← α_temp + 1
21 fin
```

seuil β, qui est le second paramètre de CPS. Ce paramètre est à configurer en fonction de la taille du cache et/ou de la bande passante du réseau. Dans la Figure 2.16, la valeur configurée de β implique le préchargement de quatre unités de présentation à chaque niveau.

Le préchargement, tributaire des paramètres α et β, se poursuit tant que la place disponible dans le cache le permet. En cas de cache plein, les unités qui feront l'objet d'un remplacement sont celles dont la valeur de pertinence est la plus faible et dont la date de chargement dans le cache est à la fois la plus ancienne et inférieure à la date courante, avec laquelle seront marquées les unités remplaçantes. La deuxième condition a été insérée afin de ne pas remplacer des unités que l'on vient de charger dans le cache.

Notons enfin que CPS travaille à un niveau plus fin que celui des unités de presentation, au niveau de blocs ou *fragments* de données. Une valeur attribuée à une unité de présentation est associée à tous les fragments appartenant à cette unité. La politique CPS intégrant le remplacement et le préchargement est présenté par l'algorithme 1.

2.3 Étude de performance

Afin d'évaluer les performances de CPS, il est indispensable de les comparer avec une autre politique. Or, compte tenu de l'originalité du parcours rapide proposé, il n'existe pas d'autre stratégie conçue pour une telle utilisation du cache. Nous avons alors considéré la politique LRU, qui est souvent employée pour la gestion des caches de lecture (voir les Sections 1.4.1 et 1.4.3.1). LRU est une heuristique de remplacement pure, qui repose sur la date du dernier accès aux fragments [3], et elle n'effectue pas de préchargement. Une comparaison entre CPS et LRU n'est donc pas équitable, puisque CPS est à la fois une stratégie de remplacement et de préchargement, c'est-à-dire une politique intégrée. Une meilleure performance affichée par CPS peut être due aux préchargements effectués. Ainsi, nous avons modifié LRU en y intégrant un préchargement « simple », au lieu du chargement à la demande par défaut. Il s'agit de précharger u unités de présentation à restituer dans le futur en cas de lecture « normale ». La version modifiée, appelée LRU-P, est ainsi une politique intégrée, effectuant du préchargement et du remplacement, au même titre que CPS. Une comparaison entre les deux politiques permet alors de mesurer l'efficacité et le coût du préchargement de CPS, qui est basé sur des heuristiques calculées à partir des fonctions de pertinence (2.9) et (2.10), par rapport à un préchargement simple.

L'évaluation de la politique CPS a été effectuée au travers de nombreuses séries d'expérimentations. Celles-ci ont été menées à l'aide d'un simulateur à événements discrets, que nous avons développé à cette fin. Nous avons confronté les deux stratégies CPS et LRU-P, selon deux critères représentants le gain apporté par une politique et le coût de son utilisation :

- **Gain :** La réduction en temps de latence due à la disponibilité des données à restituer dans le cache de lecture lors d'une interaction de parcours rapide. Cette réduction est étroitement corrélée avec le taux de succès du cache qui, rappelons le, correspond au ratio entre le nombre de fragments de données trouvés dans le cache lorsque leur restitution est demandée et le nombre total de fragments restitués (cf. Section 1.4).

3. Lors d'un accès à un fragment, dans le but de le restituer, LRU lui associe une date d'accès. Puis, en cas de cache plein, elle remplace les fragments les moins récemment restitués, c'est-à-dire les fragments auxquels on a accédé le moins récemment.

- **Coût :** Le ratio entre le nombre de fragments de données chargés/préchargés dans le cache, c'est-à-dire transmis à travers le réseau, durant la restitution d'une présentation et le nombre de fragments effectivement restitués.

Nous décrivons tout d'abord le scénario de test utilisé pour l'évaluation de CPS. Puis, dans la Section 2.3.2, nous présentons les résultats des simulations que nous avons conduites. Nous concluons cette section par une discussion de ces résultats.

2.3.1 Scénario de test

Nous avons simulé le déroulement de cinquante présentations multimédias. Les objets contenus dans ces présentations se limitent à des vidéos. Ce choix conserve la généralité des tests et se justifie par le fait que les objets médias statiques ont une taille négligeable par rapport aux médias continus comme les vidéos.

Chacune des présentations contient un certain nombre de vidéos qui se succèdent l'une à l'autre (voir Figure 2.17). Les vidéos ne se chevauchent donc pas, le début d'une vidéo correspondant exactement à la fin de la précédente, sauf pour la première vidéo. Nous avons supposé que chaque vidéo expose une idée unique et par conséquent qu'une présentation contient autant d'idées que de vidéos. Les points d'interaction dans une présentation correspondent ainsi aux instants de début des vidéos incluses dans la présentation. Le nombre de vidéos par présentation est une variable aléatoire distribuée de manière uniforme entre 10 et 30 vidéos. Le format d'une vidéo peut être MPEG-1 (1,5 Mbits/sec) ou alors MPEG-2 (4 Mbits/sec) avec une probabilité identique et égale à 1/2 pour chacun des deux formats. La durée d'une vidéo est aussi une variable aléatoire qui suit une distribution uniforme entre 20 et 30 secondes.

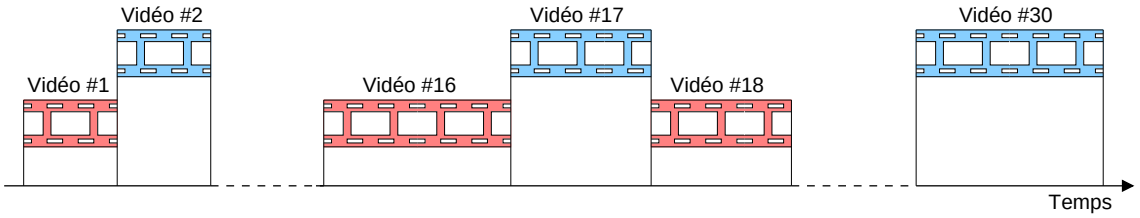


Figure 2.17 – Scénario d'une présentation de test avec 30 vidéos qui se succèdent les unes aux autres.

Les interactions des utilisateurs ont été modélisées suivant un processus de Poisson avec une fréquence d'arrivée λ, où $\frac{1}{\lambda} = \frac{\text{durée de présentation}}{x}$, $x > 0$ est un entier dont la valeur par défaut a été fixée à 10. Ceci se traduit par le fait qu'une interaction se produit toutes les $\frac{1}{\lambda}$ secondes. Plus la valeur de x est grande, plus forte est la fréquence d'arrivée d'interactions λ, et par conséquent le nombre d'interactions effectuées durant une présentation est plus important. Le type de l'interaction est, quant à lui, une des interactions suivantes : redémarrage d'une présentation, reprise d'une idée, avance rapide ou alors recul rapide, avec une probabilité identique et égale à 1/4 pour chacune de ces interactions.

Par défaut, la taille du cache a été fixée à 128Mo. Le déroulement de chaque présentation, avec les mêmes interactions qui lui sont associées, est simulé deux fois : une avec LRU-P comme politique de gestion de cache et l'autre avec CPS en tant que stratégie de gestion de cache.

Le nombre d'unités de présentation à précharger par LRU-P est fixé à 10 unités. Les valeurs respectives des paramètres α et β de la politique CPS ont été fixées à 1 et $0,95$ par défaut.

Catégorie	Paramètre	Valeur par défaut
Général	Nombre de présentations Taille de cache	50 128 Mo
Scénario	Nombre de vidéos par présentation *et donc* Nombre d'idées par présentation	Distribution uniforme (10 - 30) Distribution uniforme (10 - 30)
	Durée d'une vidéo *et donc* Durée d'une idée	Distribution uniforme (20 - 30) seconde Distribution uniforme (20 - 30) seconde
	Format des vidéos	Distribution uniforme {MPEG-1 (1,5 Mbits/s), MPEG-2 (4 Mbits/s)} (probabilité = 1/2, chacun)
Interactions	Fréquence d'interaction	Processus de poisson d'intensité λ, $\lambda = x$/Durée de présentation , $x = 10$ (une interaction se produit toutes les Durée de présentation/10 secondes)
	Type d'interaction	Distribution uniforme {redémarrage, reprise d'idée, avance rapide, recul rapide} (probabilité = 1/4, chacune)
LRU-P	Unités de présentation à précharger	$u = 10$
CPS	Seuil de préchargement β Paramètre α	$0,95$ 1

Tableau 2.1 – Récapitulatif des valeurs par défaut des différents paramètres considérés.

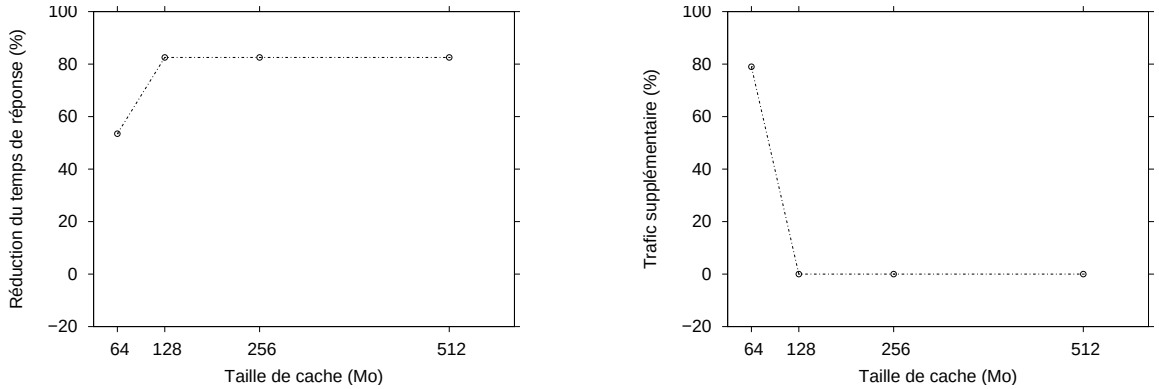
2.3.2 Résultats

En se basant sur les valeurs des paramètres listés dans le tableau 2.1, nous avons mené trois séries d'expérimentations afin de mesurer l'impact de la taille du cache, du seuil de préchargement β et du paramètre α sur les performances enregistrées par CPS. En outre, nous avons testé l'efficacité de CPS dans des environnements caractérisés par diverses niveaux d'interactivité. Les résultats de ces tests sont présentés dans les paragraphes suivants.

2.3.2.1 Impact de la taille du cache

De manière générale, les techniques de préchargement sont particulièrement sensibles à la taille du cache mis à leur disposition. Nous nous sommes donc attaché à étudier les performances de CPS en fonction de la variation de la taille du cache mis à sa disposition. Pour cela, nous

avons simulé des caches de différentes tailles, $\{64, 128, 256, 512\}$ Mo, correspondant à des caches qui peuvent être mis à la disposition des lecteurs multimédias lancés sur des terminaux mobiles de type PDA, des ordinateurs portables, etc. Pour chaque configuration, nous avons mesuré, en moyenne sur les cinquante présentations simulées, le coût et le gain apportés par le préchargement CPS par rapport à un préchargement simple. Les résultats obtenus sont représentés dans la Figure 2.18.



(a) Gain apporté par le préchargement de CPS

(b) Coût de préchargement de CPS

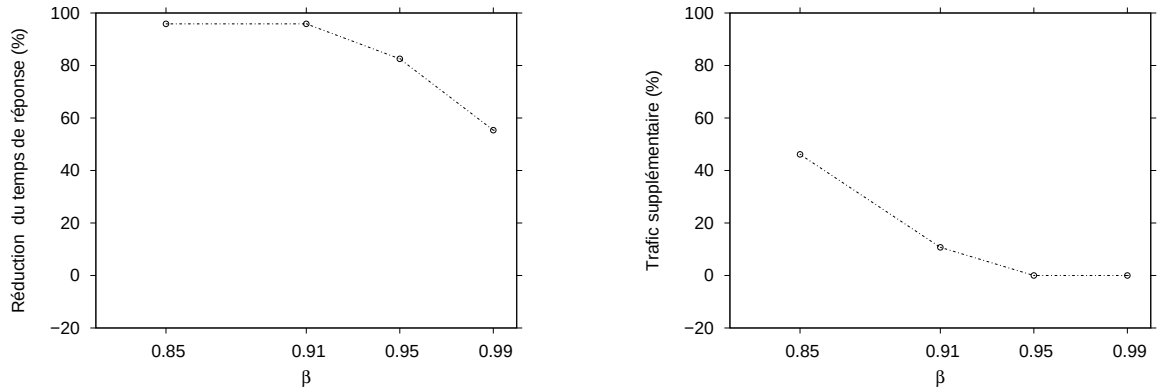
Figure 2.18 – CPS : impact de la taille du cache.

On remarque dans la figure 2.18.a qu'avec une configuration avec 64 Mo de cache, le préchargement de CPS permet de réduire le temps de réponse à une interaction du parcours rapide d'environ 50% par rapport à un préchargement simple. Cette réduction en temps de réponse devient plus prononcée et est d'environ 80% dans une configuration avec un cache de 128 Mo. Ceci s'explique trivialement par le fait que plus la taille du cache est importante, plus ce dernier est en mesure de contenir de fragments de données et par conséquent plus forte est la probabilité qu'un fragment soit présent dans le cache lorsque l'on a besoin de le restituer. Sur la même figure, on remarque également que la réduction de temps de réponse à une interaction de parcours rapide est bornée. En effet, cette réduction devient stable après un certain seuil de taille du cache (128 Mo dans les scénarios simulés). Ceci s'explique par le fait qu'à partir du moment où CPS dispose de suffisamment d'espace cache pour conserver les fragments de données auxquels on aura accès en cas d'interaction de parcours rapide, l'augmentation de la taille du cache n'a pas vraiment d'influence sur le temps de réponse à une telle interaction.

D'un autre côté, lorsque la taille du cache mis à disposition de CPS est suffisamment grande pour conserver les fragments de données auxquels on aura accès en cas d'interaction de parcours rapide, le préchargement de CPS n'induit pas de trafic supplémentaire (voir Figure 2.18.b). Elle génère le même trafic qu'un préchargement simple produit durant la présentation sauf qu'elle prévoit les interactions de parcours rapide et précharge les fragments de données avant que l'on ait besoin de les restituer. Le préchargement de CPS devient cependant très coûteux et peut générer jusqu'à 80% de trafic supplémentaire en comparaison avec un préchargement simple dans le cas de petits caches, comme on le voit sur la Figure 2.18.b. Ainsi, afin que le préchargement de CPS soit efficace, un espace cache suffisamment grand doit être mis à sa disposition. Dans la suite des expérimentations, nous utiliserons une configuration avec un cache de 128 Mo.

2.3.2.2 Impact du seuil de préchargement β

Le seuil de préchargement (β) contrôle le nombre d'unités de présentation à précharger dans chaque idée faisant l'objet d'un préchargement. Plus la valeur de β est petite, plus CPS précharge d'unités de présentation dans chaque idée à précharger. Dans cette série d'expérimentations, nous nous sommes intéressés à étudier l'impact de la valeur de β sur les performances de CPS. Afin de mesurer cet impact, nous avons fait varier la valeur de β entre $0,85$ et $0,99$. Pour chacune des valeurs, nous avons mesuré, en moyenne sur les cinquante présentations simulées, le coût et le gain apportés par le préchargement CPS par rapport à un préchargement simple. Les résultats obtenus sont reportés dans la Figure 2.19.



(a) Gain apporté par le préchargement de CPS

(b) Coût de préchargement de CPS

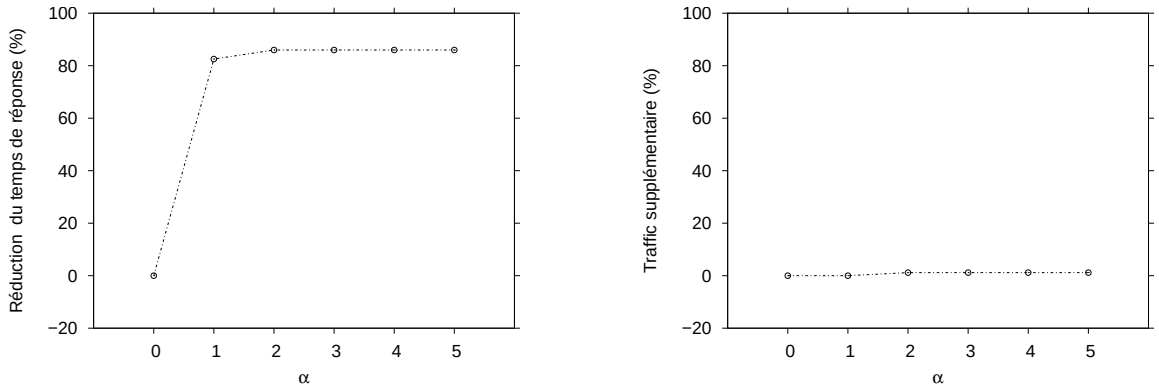
Figure 2.19 – CPS : impact du seuil de préchargement β.

Dans la Figure 2.19.a, on remarque que le gain apporté par le préchargement de CPS, par rapport à un préchargement simple, augmente avec la décroissance de la valeur attribuée au seuil de préchargement β. En effet, le préchargement de CPS permet de réduire le temps de réponse à une interaction de parcours rapide de 95% avec $\beta = 0,91$ alors que cette réduction est seulement de 55% lorsque $\beta = 0,99$. Ceci est la conséquence directe du fait que plus la valeur de β est petite, plus CPS précharge d'unités de présentation au sein des idées à précharger. Néanmoins, les opérations de préchargements sont limitées par la taille du cache, car CPS ne peut pas précharger plus de données que le cache ne peut en contenir. La stabilisation de la réduction du temps de réponse, que l'on constate sur la Figure 2.19.a, le confirme.

Par ailleurs, le trafic généré par le préchargement CPS augmente, lui aussi, avec la décroissance de β. Comme on peut le voir sur la Figure 2.19.b, le trafic supplémentaire dû au préchargement CPS, par rapport à un préchargement simple, est d'environs 45% avec $\beta = 0,85$ tandis que ce trafic supplémentaire se réduit à 10% lorsque $\beta = 0,91$ et devient nul à partir de $\beta = 0,95$. Le choix de la valeur de β devrait donc représenter un compromis entre le gain en réduction du temps de réponse et le coût de cette réduction en terme de trafic supplémentaire. Nous l'avons fixé à 0,95% ce qui permet d'améliorer le temps de réponse à une interaction de 80% tout en préservant un trafic supplémentaire nul.

2.3.2.3 Impact du paramètre α

Le paramètre α contrôle le nombre d'idées à précharger par CPS ; il indique la « profondeur » temporelle pris en compte par CPS lors du préchargement. Le nombre d'idées à précharger augmente avec la croissance de la valeur de α comme indiqué dans la Section 2.2.4.3. Nous nous sommes attachés, dans cette série d'expérimentations, à mesurer l'impact de la valeur attribuée à ce paramètre sur les performances de CPS. Pour cela, nous avons fait varier la valeur de α entre 0 et 5. Puis, nous avons mesuré pour chacune de ces valeurs, en moyenne sur les cinquante présentations simulées, le coût et le gain apportés par le préchargement CPS par rapport à un préchargement simple. Les résultats obtenus sont présentés dans la Figure 2.20.



(a) Gain apporté par le préchargement de CPS

(b) Coût de préchargement de CPS

Figure 2.20 – CPS : impact du paramètre α.

Dans la Figure 2.20.a, on constate que la performance atteinte par le préchargement CPS avec $\alpha = 0$ est équivalente à celle atteinte par un préchargement simple. En effet, la stratégie CPS se transforme en stratégie de préchargement simple lorsque $\alpha = 0$. On peut également observer sur la même figure qu'à partir de $\alpha = 1$ la réduction du temps de réponse apportée par le préchargement CPS, par rapport à un préchargement simple, tend vers une valeur fixe (80%), et ce quelle que soit la valeur attribuée au paramètre $\alpha \geq 1$. Ceci s'explique par le fait que les interactions du parcours rapide proposé sont liées à l'idée en cours de restitution. Les idées qui ne succèdent (ou ne précédent) pas immédiatement l'idée courante ne sont pas immédiatement accessibles par une interaction de parcours rapide et par conséquent n'ont pas d'impact sur le temps de réponse à une telle interaction.

En outre, le trafic supplémentaire du préchargement CPS, par rapport à un préchargement simple, reste nul, quelle que soit la valeur attribuée au paramètres α (voir Figure 2.20.b). Ceci n'est pas uniquement lié au paramètres α mais à l'ensemble des paramètres considérés par nos simulations et dont les valeurs par défaut sont récapitulées dans le tableau 2.1. En particulier, la taille du cache choisie permet à CPS de réaliser ses préchargements sans effectuer pour autant des remplacements inutiles. À l'égard de ces résultats, nos concluons que $\alpha = 1$ est une valeur tout à fait convenable pour être attribuée à ce paramètre.

2.3.2.4 Impact de l'interactivité des utilisateurs

Dans cette série d'expérimentations, nous nous sommes intéressés à mesurer la performance de CPS dans des environnements caractérisés par des niveaux d'interactivité divers. Pour cela, nous avons fait varier la fréquence d'arrivée des interactions de parcours rapide, de la part des utilisateurs, en faisant varier le temps moyen écoulé entre deux interactions successives. Ce temps a évolué en fonction de la durée d'une présentation comme suit :

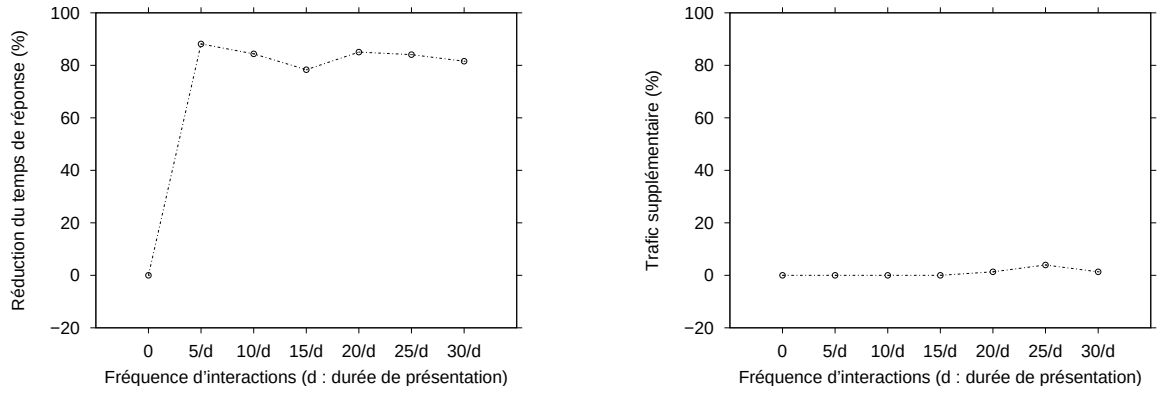
$$\text{Temps moyen écoulé entre deux interactions successives} = \frac{\text{Durée de présentation}}{x}\,,\ x > 0$$

Ainsi, une interaction de parcours rapide arrive toutes les $\frac{\text{Durée de présentation}}{x}$ secondes , $x > 0$ et la fréquence d'arrivée d'interactions s'exprime donc par :

$$\text{Fréquence d'arrivée d'interactions : } \lambda = \frac{x}{\text{Durée de présentation}}\,,\ x \in \{5, 10, 15, 20, 25, 30\}$$

Une fréquence d'arrivée nulle signifie qu'il n'y a pas d'interaction de parcours rapide.

Nous avons ensuite mesuré, en moyenne sur les cinquante présentations simulées, le coût et le gain apportés par le préchargement CPS par rapport à un préchargement simple dans chacune des configurations, qui correspondent à des environnements plus ou moins interactifs. Les résultats obtenus sont présentés dans la Figure 2.21.



(a) Gain apporté par le préchargement de CPS

(b) Coût de préchargement de CPS

Figure 2.21 – CPS : impact de la fréquence des interactions.

Dans la Figure 2.21.a, on constate qu'en l'absence d'interaction de parcours rapide (fréquence d'interaction nulle) l'emploi de CPS ne change rien. En revanche, dans des environnements interactifs (fréquence d'interactions > 0) le préchargement de CPS permet une nette amélioration des performances, en terme de réduction du temps de réponse à une interaction de parcours rapide, en comparaison avec un préchargement simple. En effet, on note sur la même figure que cette amélioration est stable et d'environ 80%, quelle que soit la fréquence d'arrivée des interactions.

Enfin, il est intéressant de noter que le trafic supplémentaire généré par le préchargement de CPS, par rapport au trafic produit par un préchargement simple, est nul quelle que soit la fréquence d'arrivée des interactions (voir Figure 2.21.b). Ceci est dû notamment à deux

raisons : d'une part, la taille du cache utilisé permet à CPS de réaliser ses préchargement sans effectuer pour autant des remplacements inutiles, et d'autre part, la valeur attribuée au seuil de préchargement a été choisie de manière à assurer un bon compromis entre le gain apporté par le préchargement de CPS (en terme de réduction de temps de réponse à une interaction) et le trafic supplémentaire qui en résulte par rapport à un préchargement simple.

2.3.3 Discussion

Les séries de tests présentés dans la section précédente, nous ont permis de quantifier les paramètres de CPS α et β d'une part, et de mesurer ses performances en fonction de différents facteurs d'autre part. Ces facteurs concernent particulièrement la taille du cache de lecture et l'interactivité des utilisateurs.

À partir du moment où le cache mis à disposition de CPS est suffisamment grand pour effectuer ses préchargements (au delà de 128 Mo dans notre scénario de test), la pertinence de CPS est remarquable. Le gain, en terme de réduction du temps de réponse à une interaction, apporté par le préchargement de CPS par rapport à un préchargement simple est d'environ 80%. D'ailleurs, ce préchargement n'induit pas de trafic supplémentaire. En revanche, si l'espace cache est insuffisant pour conserver les fragments de données auxquels on aura accès en cas d'interaction de parcours rapide, ce préchargement devient coûteux et le trafic supplémentaire peut augmenter jusqu'à 80% par rapport à un préchargement simple. Donc, pour que CPS réalise de bonnes performances, un des points clefs est de lui fournir un cache de taille suffisamment grande pour effectuer ses préchargements. Cette taille dépend de la présentation elle même mais aussi des paramètres de CPS : α et β.

Au vu des résultats des expérimentations, nous avons pu quantifier les paramètres α et β. Il convient d'attribuer une valeur $\alpha = 1$ à ce paramètre qui contrôle le nombre d'idées à précharger. Les idées associées à des valeurs de $\alpha > 1$, ne sont pas accessibles via une interaction de parcours rapide et par conséquent le fait de les précharger dans le cache n'a pas d'impact sur le temps de réponse à de telles interactions. Quant au seuil de préchargement, il est tributaire de la taille du cache mis à disposition de CPS. Des petites valeurs de β peuvent générer un trafic supplémentaire important par rapport à un préchargement simple lorsque l'espace cache se trouve limité. Ce paramètre doit donc être configuré avec prudence. Dans nos simulations, nous avons constaté que $\beta = 0,95$ permet d'obtenir un bon compromis entre le gain apporté par le préchargement de CPS par rapport à un préchargement simple et le trafic supplémentaire qui en résulte. Cette valeur permet d'obtenir une réduction de 80% du temps de réponse à une interaction de parcours rapide, tout en assurant un trafic supplémentaire nul.

Enfin, les résultats obtenus par les expérimentations que nous avons menées montrent clairement la pertinence et l'efficacité du préchargement CPS quelle que soit la fréquence d'arrivée des interactions de parcours rapide. D'ailleurs, les performances observées du préchargement CPS sont stables et permettent une nette réduction (environ 80%) des temps de réponse à des interactions de navigation accélérée par sauts temporels dans des présentations multimédias par rapport à un préchargement simple, et ce même dans des environnements hautement interactifs. En plus, le trafic supplémentaire généré par le préchargement de CPS est nul par rapport à un préchargement simple. Notons finalement que ces bonnes performance sont étroitement liées à deux facteurs : la taille du cache dont dispose CPS pour effectuer ses préchargements et le seuil de préchargement β.

2.4 Conclusion

La navigation accélérée dans les présentations multimédias ne peut être effectuée en utilisant la méthode employée pour accélérer des vidéos. Ceci est dû à de nombreux problèmes, aussi bien techniques que sémantiques, qui font obstacle à la mise en œuvre d'une telle accélération. Le présent chapitre a constitué une contribution à la résolution de ces problèmes.

Après avoir étudié de près la navigation accélérée de documents multimédias, nous avons remarqué que celle-ci est étroitement liée à leur contenu. Nous avons donc proposé une nouvelle sémantique du parcours rapide des présentations multimédias. Cette sémantique constitue une solution aux problèmes conceptuels qui empêchaient la mise en pratique de la navigation accélérée dans les présentations multimédias. Elle est spécialement conçue pour parcourir rapidement des présentations multimédias tout en conservant la compréhension humaine de ces dernières. Il s'agit d'effectuer des sauts dans l'axe du temps d'une présentation plutôt que de l'accélérer. L'avantage de notre approche découle du fait que ces sauts temporels ont été reliés au contenu du document consulté.

Dans le cadre d'accès aux présentations multimédias en streaming, le parcours rapide que nous avons proposé se trouve pénalisé par le fait qu'un usager doit attendre que le lecteur de présentations rapatrie les données à restituer en réponse à une interaction de parcours rapide. Compte tenu de la taille des données audiovisuelles, couplées avec le nombre d'objets que peut contenir une présentation, le temps de ce rapatriement peut s'avérer relativement long et un usager peut décider de se rétracter suite à une telle attente. Afin de réduire le temps de réponse à des interactions de parcours rapide, nous avons fait appel à une technique de préchargement de données avant que l'on ait besoin de les restituer. Cette technique, spécialement conçue pour le parcours rapide proposé, a été élaborée au sein d'une politique de gestion du cache de lecture, du côté client. Cette politique, appelée CPS, est une politique intégrée. C'est à la fois une stratégie de remplacement et de préchargement. Nous avons effectué plusieurs séries d'expérimentations afin de valider la pertinence de CPS. Les résultats obtenus montrent clairement qu'elle permet d'obtenir de bonnes performances même dans des environnements hautement interactifs, notamment lorsque la taille du cache mis à sa disposition est suffisamment grande et le seuil de préchargement est judicieusement configuré.

Le chapitre suivant est consacré à une autre problématique dont souffrent certains systèmes de streaming, à savoir la disponibilité des données au sein de ces systèmes.

* * *

Chapitre 3

Disponibilité des données dans les systèmes de streaming P2P

Jusqu'à présent, nous avons considéré les systèmes de streaming « traditionnels ». Ces systèmes sont composés d'un serveur dédié à la tâche du streaming de contenus multimédias stockés dans son espace de stockage, de proxies agissants à la fois comme passerelles et sous-serveurs et enfin de clients chargés de restituer les contenus aux utilisateurs. Toutefois, comme nous l'avons déjà mentionné dans la Section 1.5, les systèmes de streaming ont évolué ces dernières années. Les serveurs, autrefois hébergés par des machines massivement parallèles, se sont fait concurrencés tout d'abord par des grappes de machines (serveurs parallèles). Ensuite, ces serveurs ont été transposés sur des grilles de machines (serveurs sur grille), faisant ainsi disparaître les proxies des systèmes de streaming. Plus récemment, la technologie P2P a laissé ses marques sur ces systèmes et elle est arrivée à faire disparaître, en plus des proxies, les serveurs de certaines architectures. Les systèmes de streaming basés sur la technologie P2P sont appelés *systèmes de streaming P2P* et constituent le centre d'intérêt de ce chapitre. Ce genre de système a émergé notamment grâce à la multiplication des accès haut-débit à Internet *via* des machines ayant de plus en plus de puissance de traitement et de stockage.

Dans les systèmes de streaming P2P, composés uniquement de *pairs*, ces derniers prennent en charge les tâches du serveur et des proxies. Les pairs dans de tels systèmes effectuent donc, en plus de la restitution de contenus multimédias, le streaming et la retransmission de ces contenus. Les systèmes de streaming P2P permettent aux utilisateurs de *partager* leurs contenus multimédias en mode streaming, combinant ainsi les avantages de la technologie P2P et du streaming. Toutefois, ces systèmes sont caractérisés par la volatilité des usagers. En effet, un utilisateur peut quitter le système à n'importe quel moment et en particulier durant le streaming d'un contenu qu'il partage. Cette volatilité entraîne un problème de disponibilité des contenus au sein du système. Le départ d'un utilisateur partageant un contenu en cours d'accès par un autre empêche ce dernier d'en terminer le visionnage.

La disponibilité des données, au sein des systèmes de streaming P2P composés uniquement de pairs, reste à ce jour une problématique ouverte. Afin de contribuer à une solution à ce problème, nous avons proposé d'utiliser un cache, centralisé dans un premier temps, pour préserver certaines parties de contenus multimédias en cours d'accès. Plutôt que de dupliquer la totalité des données, nous avons opté pour la préservation des *suffixes* des contenus qui sont en cours d'accès et qui

seront vraisemblablement restitués jusqu'à leur fin. Ce choix permet, comme l'ont confirmés les résultat de nos simulations, d'améliorer la disponibilité des données au sein du système tout en générant un minimum de trafic supplémentaire. Pour des raisons de faisabilité, notre objectif n'est donc pas d'assurer une disponibilité de toutes les données, mais plutôt de limiter les effets négatifs du départ imprévisible des pairs sur celle-ci, avec un coût raisonnable. Notre approche a été fondée sur la définition d'un modèle probabiliste permettant de calculer la taille des suffixes à mettre en cache, en fonction de l'avancée des lectures dans les différents médias d'une part, et sur la proposition d'une gestion de cache associée d'autre part.

Dans un deuxième temps, et afin de s'affranchir de la contrainte que constitue un cache centralisé dédié, nous nous sommes intéressés à une architecture distribuée, constituée de l'agrégation de plusieurs « petits » espaces mis à disposition par les pairs. Cette agrégation sera vue comme un *cache virtuel distribué* (DVC). Même si la gestion d'un tel cache ne diffère que peu de celle d'un cache centralisé. Puisqu'une couche logicielle peut prendre en charge l'abstraction nécessaire pour sa gestion, d'autres problèmes surviennent et notamment celui de la dynamicité de ce cache. En effet, le départ d'un pair entraîne la diminution de la taille du cache et *vice versa*. Ce changement continuel de la taille du cache impose un ajustement dynamique des différents paramètres (taille des suffixes, charge des pairs, etc.). Nous avons mis en place un mécanisme de « monitoring », fonctionnant par retour sur performance et permettant d'adapter le volume global des suffixes à mettre en cache en fonction de l'espace disponible.

Afin d'évaluer la pertinence de la mise en cache des suffixes, nous avons conduit des séries d'expérimentations avec des configurations variées. Les résultats obtenus font clairement ressortir l'efficacité de notre approche pour réduire les effets du départ des pairs sur la disponibilité de données au sein d'un système de streaming P2P. En effet, dans les simulations effectuées, on a pu observer des améliorations de la disponibilité des données allant jusqu'à 38%, pendant que le trafic supplémentaire induit par nos mécanismes de mise en cache reste acceptable.

Ce chapitre est organisé comme suit : la Section 3.1 expose les différentes facettes d'un système pair-à-pair. Nous donnons dans la Section 3.2 le cadre général de la contribution du présent chapitre, à savoir les systèmes de streaming P2P. Ensuite, dans la Section 3.3, nous décrivons la problématique principale autour de laquelle s'articule ce chapitre. Nous détaillons aussi notre approche pour y répondre, en l'occurrence *la mise en cache des suffixes*. La pertinence de cette approche est montrée par les résultats de l'étude expérimentale, présentée dans la Section 3.4.

3.1 Systèmes pair-à-pair

De manière générale, le terme pair-à-pair fait référence à une classe de systèmes qui utilisent des ressources distribuées (puissance de traitement et/ou espace de stockage) afin d'effectuer une ou plusieurs tâches de manière décentralisée [MKL+02]. Un tel système est constitué de plusieurs entités, souvent identiques, qui coopèrent sur un pied d'égalité pour effectuer une tâche donnée. Ces entités sont communément appelées *pairs* et les systèmes composés de pairs sont dénommés *systèmes égal-à-égal* ou *pair-à-pair*, traductions de l'anglais « Peer-to-Peer », qui est souvent abrégé par le sigle P2P.

Dans cette section, nous donnons une définition du concept pair-à-pair. Nous décrivons également les caractéristiques des systèmes P2P. Ensuite, nous présentons une taxinomie des différentes architectures utilisées pour mettre en œuvre de tels systèmes, ainsi que les techniques de base utilisées pour leur organisation. En guise de conclusion, nous dressons un aperçu des domaines d'application du concept P2P.

3.1.1 Définitions

Au sein d'un système dit *client-serveur* on distingue deux types d'entités : le serveur et le(s) client(s). Par entité, nous entendons un logiciel hébergé sur une machine, appelée également *nœud*. Dans un tel système, un ou plusieurs clients peuvent se connecter sur un serveur central afin de lui demander d'effectuer un *service*. Un système pair-à-pair peut être vu comme une extension des systèmes client-serveur où les nœuds remplissent à la fois les fonctions de client et de serveur. Un système pair-à-pair n'instaure donc pas de hiérarchie entre les nœuds, c'est-à-dire que tout nœud peut effectuer les même fonctionnalités. C'est là l'origine du terme *pair* utilisé pour baptiser les entités, qui sont à niveau égal, dans un système pair-à-pair.

De nombreuses définitions existent concernant le concept de pair-à-pair. Certaines le définissent comme une infrastructure dans laquelle tous les nœuds peuvent potentiellement avoir les mêmes fonctions, à la différence du modèle client-serveur [Sin01]. D'autres définitions présentent le pair-à-pair comme un système permettant de faire communiquer entre eux les différents nœuds d'un réseau sans passer par un intermédiaire [Gro08]. Dans la suite de ce document, nous utiliserons une définition générale qui rassemble de manière cohérente les principaux points des autres définitions. Elle émane de Schollmeier dans [Sch01] :

Système pair-à-pair : *Un système distribué est qualifié de pair-à-pair si les nœuds de ce système partagent une partie de leurs ressources matérielles (puissance de calcul, espace disque, interface réseau, imprimante, etc.). Ces ressources partagées sont nécessaires pour fournir les services et les contenus offerts par le système. Ces ressources sont accessibles directement par les autres pairs sans passer par des entités intermédiaires. Les participants à ce genre de système sont à la fois des fournisseurs de ressources (services et contenus) et des utilisateurs de ces ressources.*

Cependant, dans le concept de pair-à-pair, il est possible d'effectuer certaines actions de manière centralisée. Pour cette raison, Schollmeier propose une classification des systèmes pair-à-pair, composée de deux sous-domaines. Deux définitions complémentaires sont donc ajoutées à la première :

Système pseudo pair-à-pair : *Un système distribué peut être qualifié de « pseudo pair-à-pair » s'il est un système pair-à-pair et si, en plus, une entité centrale est nécessaire pour fournir une partie des services offerts par ce système.*

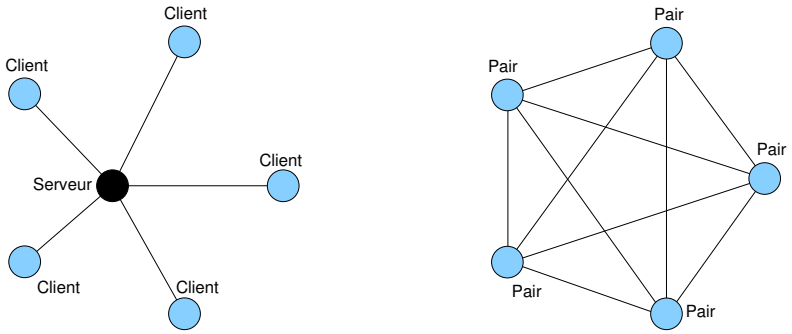
Système pair-à-pair pur : *Un système distribué peut être qualifié de « pair-à-pair pur » s'il est un système pair-à-pair et si, en plus, on peut supprimer de façon arbitraire n'importe quel nœud de ce système sans qu'il y ait dégradation ou perte de services offerts par ce système.*

3.1.2 Caractéristiques

3.1.2.1 Décentralisation

Le fonctionnement des systèmes P2P n'est pas centralisé dans une relation maître-esclave, comme c'est habituellement le cas dans les systèmes baptisés client-serveur, mais décentralisé entre les différents pairs, qui sont à la fois client et serveur des autres pairs. Ils sont aussi des passerelles, en transférant les données vers le(s) pair(s) destinataire(s). De cette manière, tous les pairs sont capables de communiquer directement entre eux, comme l'illustre la Figure 3.1(b) qui représente cinq pairs tous connectés entre eux.

Cette décentralisation présente l'avantage de réduire, voire de supprimer complètement, tout risque de goulot d'étranglement, très fréquent dans les systèmes adoptant le modèle client-serveur, comme l'illustre la Figure 3.1(a). Cependant, une architecture complètement décentralisée peut être difficile à mettre en œuvre, puisqu'il n'existe pas d'entité possédant une vue globale de tous les pairs du système. De plus, cette décentralisation peut parfois présenter certaines limitations, notamment en terme de performances lors de la recherche de ressources.



(a) Modèle client-serveur. (b) Modèle pair-à-pair.

Figure 3.1 – Vue simplifiée du modèle client-serveur versus le modèle pair-à-pair.

3.1.2.2 Extensibilité

Une conséquence directe de la décentralisation, inhérente aux systèmes pair-à-pair, est l'augmentation du nombre de ressources qu'il est ainsi possible de fédérer. Le nombre de nœuds composant le réseau n'est donc plus limité par une centralisation forte, comme dans le modèle client-serveur. Néanmoins, l'un des enjeux les plus importants dans les systèmes P2P consiste à pouvoir assurer de bonnes performances au système, notamment lors de la recherches de

ressources, lorsque le nombre de nœuds est en forte augmentation. On appelle cette caractéristique l'*extensibilité* ou encore le *passage à l'échelle*, mais on utilise plus souvent le terme anglais *scalability*.

3.1.2.3 Hétérogénéité

Le fait de fédérer un nombre si important de ressources entraîne une grande hétérogénéité des machines, tant au niveau matériel (puissance et type de processeur, débit des connexions, etc.) que logiciel (systèmes d'exploitations, etc.). Cette hétérogénéité ne doit pas pénaliser le système, dont l'un des objectifs est de maintenir une exécution efficace, quelles que soient les conditions de son déploiement. Il faut donc que le système soit portable afin de pouvoir s'exécuter sur toutes les machines de l'infrastructure, sans présenter de problème de compatibilité entre les pairs.

3.1.2.4 Tolérance aux pannes

Dans le modèle client-serveur, si le serveur tombe en panne ou devient inaccessible par le réseau, les clients seront bloqués. Dans un réseau pair-à-pair, si un pair tombe en panne, il reste toujours les autres pairs pour assurer les services offerts par ce dernier. Cet avantage fait des systèmes P2P des outils de choix pour décentraliser des services, qui doivent assurer une haute disponibilité tout en entraînant de faibles coûts d'entretien.

3.1.2.5 Volatilité

Les infrastructures P2P sont des environnements très dynamiques, puisque les pairs peuvent rejoindre ou quitter le système à n'importe quel moment, on parle aussi de nœuds « volatil ». Dans les systèmes mettant en œuvre le modèle client-serveur, la connexion ou la déconnexion d'un nœud ne concerne que le serveur, qui devra mettre à jour la liste des clients connectés. Or, dans les systèmes P2P, de part les connexions établies entre certains nœuds, l'apparition et/ou la disparition d'une ressource doivent être détectées et prises en compte. Il faut donc que le système définisse une stratégie pour que les nœuds de l'infrastructure puissent s'auto-organiser, lors de la connexion (ou de la déconnexion) d'un pair.

Par exemple, si les ressources partagées sont des fichiers, la déconnexion du nœud fournisseur entraînera automatiquement l'arrêt du téléchargement sur le demandeur. Ce dernier doit donc trouver un autre fournisseur disposant de ce même fichier, ou attendre que le pair initialement utilisé se reconnecte au système dans le cas où aucun autre pair n'est détenteur de cette ressource. Quel que soit le cas de figure, la reprise du téléchargement s'effectuera facilement à partir de l'état du fichier juste avant la déconnexion, puisque les données ont été inscrites régulièrement sur le disque dur du demandeur lors des téléchargements précédents.

En revanche, dans le cas où les ressources partagées représentent de la puissance de traitement, il faut aussi définir des mécanismes de tolérance aux pannes afin de détecter les disparitions de ressources et de désigner un nouveau pair pour effectuer les services assurés auparavant par un nœud venant d'être déconnecté.

3.1.3 Taxinomie des architectures

Comme nous l'avons vu dans la Section 3.1.1, R. Schollmeier propose deux types d'architectures possibles pour les systèmes pair-à-pair : pseudo P2P et pur P2P. En pratique, ces deux architectures se distinguent selon le fonctionnement des recherches de contenus ou de services au sein du système [LCP+05].

3.1.3.1 Architectures pseudo pair-à-pair

Les architectures pseudo pair-à-pair sont considérées comme la première génération de réseaux P2P. Elles sont conceptuellement très proches des architectures de type client-serveur. Dans ce type d'architectures, chaque pair publie ses ressources, c'est-à-dire les fichiers qu'il propose ou encore la puissance de traitement qu'il peut fournir, sur un unique nœud stable. Il s'agit d'un serveur d'annuaire qui indexe toutes les ressources de l'infrastructure. Par souci de concision, nous appelons ce serveur d'annuaire tout simplement *annuaire*, dans le reste de ce manuscrit.

Lorsqu'un nœud du système, le pair demandeur, recherche une ressource d'un certain type, il envoie une requête à l'annuaire qui choisira dans sa liste le pair le plus approprié pour satisfaire la demande : le pair fournisseur. L'annuaire retourne ensuite les informations de localisation du pair fournisseur au pair demandeur, afin que ce dernier puisse établir une connexion directe avec le pair fournisseur. Ce modèle peut, bien souvent, s'étendre dans le cas où il existe plusieurs pairs fournisseurs. Dans ce concept, le service de recherche est centralisé, mais les communications sont décentralisées.

Un exemple d'architecture de ce type est schématisé par la Figure 3.2. Ce schéma présente le cas de figure où cinq pairs (numérotés de 1 à 5) sont connectés à un annuaire central. La figure représente par des traits pointillés les requêtes de recherche de ressources ainsi que la publication de ces dernières. On voit également apparaître en traits pleins les connexions directes établies entre les pairs demandeurs et les pairs fournisseurs.

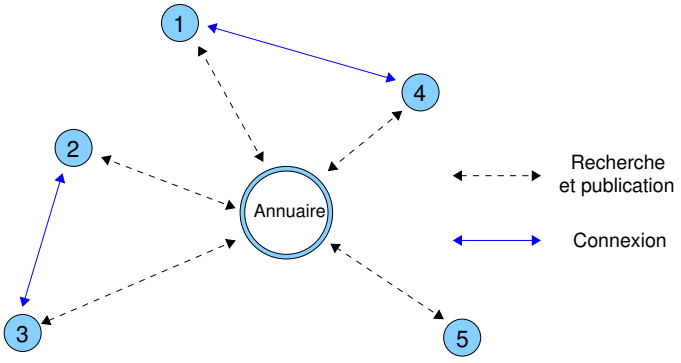


Figure 3.2 – Exemple d'architecture pseudo pair-à-pair.

Dans ce modèle d'architectures, le serveur possède une connaissance globale des nœuds du système. Cette forte centralisation limite l'extensibilité de ce dernier à cause des goulots d'étranglement qu'elle entraîne. En outre, plus le nombre de pairs connectés au système est important, plus la capacité de stockage de l'annuaire devra être accrue pour indexer les ressources. De la même manière, une augmentation du nombre de requêtes effectuées devra entraîner une augmentation de la puissance de traitement de l'annuaire.

Afin de remédier à ce problème et de permettre une meilleure extensibilité du système, des protocoles comme FastTrack [LKR06] utilisent une fédération de nœuds d'indexation, appelés *super-nœuds*, pour héberger l'annuaire qui est ainsi réparti entre ces super-nœuds. Bien que les nœuds et les super-nœuds aient une relation d'égal à égal en ce qui concerne leur mise en relation, tel n'est plus le cas pour la recherche de ressources. En effet, dans ce type d'architecture, les pairs se rattachent lors de leur utilisation à un super-nœud qui agit alors comme un mini annuaire.

Une fois connecté à leurs super-nœuds respectifs, les pairs y publient leurs ressources. Un super-nœud est donc en possession de la liste de toutes les ressources des pairs qui sont rattachés à lui, ainsi que ses propres ressources. Tous les super-nœuds sont alors interconnectés comme le représente la Figure 3.3. Lorsqu'un pair recherche une ressource, il demande à son nœud d'indexation de la localiser. Ce dernier commence alors par vérifier s'il en est détenteur ou si elle apparaît dans sa liste des ressources partagées par les nœuds rattachés à lui. S'il la trouve, il envoie la référence du détenteur au pair demandeur et la connexion s'établira de manière directe entre les deux nœuds. Dans le cas contraire, il interroge les autres super-nœuds.

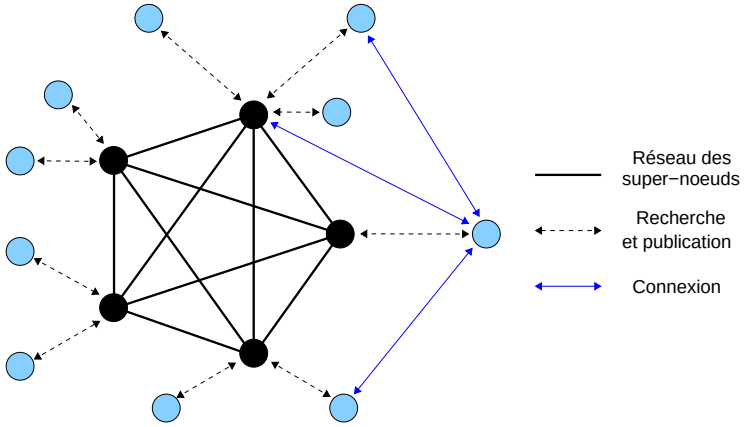


Figure 3.3 – Exemple d'architecture pseudo pair-à-pair, où l'annuaire est réparti entre plusieurs super-nœuds.

3.1.3.2 Architectures pur pair-à-pair

Dans les architectures de type pur pair-à-pair, il n'existe aucune entité centrale permettant d'indexer les pairs connectés ainsi que leurs ressources. Le système fonctionne de manière totalement distribuée et permet donc de gérer efficacement le caractère dynamique des pairs. En d'autres termes, l'annuaire n'est plus sur un seul nœud, ni réparti entre en un ensemble restreint de super-nœuds, mais plutôt distribué sur tous les nœuds du système.

La première action à réaliser par un nœud est l'amorçage [1] : pour se connecter au réseau, un pair balaye le réseau (local, Internet, etc.) à la recherche d'autres nœuds. Lorsqu'un pair est trouvé, un lien est créé entre les deux nœuds qui deviennent alors voisins. Ce mécanisme se produit jusqu'à ce que le pair considéré ait un nombre suffisant de liens. L'ensemble des pairs et des liens forment un graphe (cf. Figure 3.4) qui constitue ainsi un *réseau logique*, aussi appelé *overlay network* dans la littérature anglophone. On distingue alors les *réseaux non structurés*,

1. Dans la littérature, on utilise parfois le terme anglais *bootstrapping*

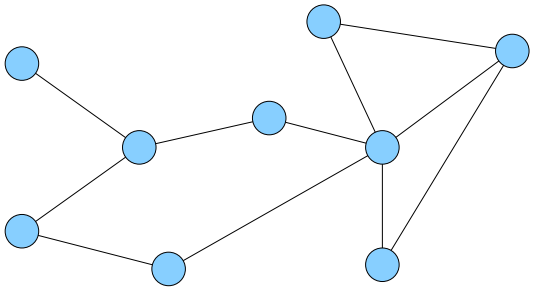


Figure 3.4 – Exemple d'un graphe formé par l'interconnexion des pairs.

où les recherches se font de proche en proche à travers le réseau, et les *réseaux structurés*, où il existe une organisation automatique qui facilite la recherche de ressources.

Notons que la stratégie d'amorçage effectuée en balayant le réseau peut être une technique envisageable au sein d'un réseau local, mais semble peu réalisable à l'échelle de l'Internet. Une solution simple à ce problème consiste à ce que chaque pair désirant rejoindre l'infrastructure connaisse au moins un de ses membres. Grâce à ce dernier, le nouveau pair peut ensuite se connecter aux autres nœuds et élargir ainsi son voisinage. Dans la pratique, cette stratégie peut être mise en place en utilisant une liste des pairs, qui sont supposés être toujours, ou assez souvent, connectés et constituant *de facto* le cœur de l'infrastructure.

Réseaux non structurés

Dans les réseaux non structurés, chaque pair est en fait responsable de l'indexation de ses propres ressources. Un pair qui cherche une ressource diffuse une requête à ses voisins. Si la ressource attendue n'est pas présente sur ces derniers, la requête est alors propagée à leurs voisins respectifs, et ainsi de suite. Le protocole agit donc par *inondation* des nœuds du graphe ; on utilise également le terme anglais *flooding* pour décrire cette propagation d'information, ou de requêtes, de proche en proche sur les pairs de l'infrastructure.

Si un pair recevant la requête possède effectivement la ressource recherchée, il ne propage plus l'information à ses voisins et envoie une réponse à destination de l'émetteur. La réponse suit alors le parcours inverse de la requête en se propageant de pair en pair jusqu'au destinataire. Il faut donc que chaque pair propageant un message retienne l'origine et la destination de celui-ci lors de son passage. Par la suite, la connexion entre le demandeur et le fournisseur peut alors s'établir directement ou bien au travers des pairs intermédiaires utilisés au préalable pour propager la requête.

Afin de limiter l'inondation et les boucles, chaque requête possède une durée de vie, notée TTL pour *Time To Live*. Ce mécanisme est en fait mis en place grâce à un compteur de sauts, qui est décrémenté à chaque fois qu'un pair propage la requête. Ce modèle part de l'observation qu'un message peut atteindre n'importe quel point d'un réseau après un petit nombre de sauts, effectué chacun au hasard. Lorsque la durée de vie d'une requête arrive à terme ($TTL = 0$), la requête est détruite, c'est-à-dire qu'elle n'est plus propagée.

Toutefois, ce fonctionnement totalement décentralisé et déstructuré, aussi philosophiquement plaisant soit-il, possède plusieurs inconvénients. D'abord, le manque d'efficacité des recherches, puisque tous les nœuds du réseau ne peuvent être interrogés. Ensuite, la latence importante due

à la longueur de la propagation des recherches et au retour de celles-ci vers le pair l'ayant initié. Enfin, le nombre de requêtes croit de manière exponentielle avec le nombre de pairs connectés au système. Ceci est notamment dû au mode de communication utilisé pour le transfert de requêtes, qui est de type *broadcast* et pour lequel chaque nœud envoie une requête à l'ensemble de ses voisins. Cet inconvénient a donc pour effet d'encombrer le réseau de communication et peut même le saturer, rendant ainsi tout échange impossible entre les pairs.

Réseaux structurés

Afin d'abaisser la complexité de la recherche des ressources au sein des systèmes P2P, les concepteurs de ceux-ci se sont tournés vers des structures de données bien connues pour leur efficacité, à savoir *les tables de hachage*. Il s'agit d'une table de couples `(clé, identifiant)` permettant de récupérer, et de stocker, l'identifiant du pair détenteur d'une ressource donnée, celle-ci étant identifiée par la clé. La clé ainsi que l'identifiant sont obtenus en utilisant une fonction de hachage [2] sur, respectivement, une chaîne de caractères identifiant la ressource de manière univoque et l'adresse IP du pair détenteur de cette ressource. Ces couples sont répertoriés par tous les nœuds du système, de telle sorte que la déconnexion d'un pair n'entraîne pas de pertes importantes pour l'ensemble du réseau, on parle alors de *table de hachage distribuée* (DHT). Certains algorithmes de recherche dans la DHT, comme Chord [SMK+01], peuvent avoir une complexité pouvant atteindre au maximum $O(log(n))$ pour un réseau constitué de n nœuds. Ceci assure théoriquement un temps de réponse à une requête plus court que dans les réseaux non structurés.

L'utilisation de DHT permet de limiter le nombre d'envois de requêtes concernant la recherche de ressources, puisqu'il n'est pas nécessaire d'envoyer un message à tous, mais simplement de transmettre la requête de pair en pair. Cette recherche continue jusqu'à trouver le pair possédant l'information recherchée en se rapprochant de lui à chaque « saut » dans le réseau. Une notion de *distance virtuelle* est ainsi introduite, sous forme d'une fonction calculant la distance entre deux nœuds.

En s'appuyant sur la notion de distance virtuelle, les pairs sont organisés de telle sorte que la distance est minimisée entre deux nœuds voisins. Cette distance peut être fonction de la disposition géographique des nœuds comme dans CAN [RFH+01], de la latence de la connexion comme dans Pastry [RD01a] et Tapestry [ZHS+04] ou alors définie à l'aide d'un opérateur mathématique, tel que le « OU exclusif », qui est aussi appelé XOR, comme dans Kademlia [MM02].

L'espace d'adressage des clés (*Keyspace*) est réparti entre les pairs de manière à ce qu'un pair possède toutes les clés dont son identifiant est le plus proche, au sens de la distance virtuelle adoptée. L'intérêt principal de cette répartition est que l'ajout ou la suppression de nœuds change uniquement les clés des nœuds adjacents et laisse tous les autres nœuds inchangés. Notons qu'il est d'usage d'introduire de la redondance dans la DHT afin que le départ d'un nœud du système n'engendre pas la perte de la localisation des ressources qu'il indexe et ne rende impossible l'accès à ces ressources.

2. La fonction de hachage la plus couramment utilisée est la fonction SHA-1, fonction utilisée en cryptographie et par exemple dans des protocoles sécurisés comme SSH.

3.1.4 Domaines d'applications

Au vu de la définition du concept P2P, il est clair que ce modèle présente une alternative viable au concept de client-serveur. Il s'agit de partager des ressources entre les différents nœuds du système, où chaque nœud peut simultanément donner et obtenir de ces ressources. Ceci diffère fondamentalement du concept de client-serveur, où un seul nœud (le serveur) donne des ressources aux autres. Le modèle P2P n'est cependant pas récent, puisque il remonte à la fin des années 60, au début même de l'Internet [DFM01]. En effet, le tout premier réseau à transfert de paquets ARPANET, qui connectait plusieurs centres de recherche à travers les États-Unis, afin de partager leur puissance de traitement et les données qu'ils détiennent, était basé sur le principe du P2P. Par la suite, et notamment avec la démocratisation des ordinateurs personnels et la vulgarisation des interconnexions réseaux et tout ce qui en découle comme problèmes de sécurité de données, le modèle client-serveur est devenu le modèle prédominant de l'Internet. Cependant, au cours de l'année 1999, la suprématie de ce modèle a décliné au profit du modèle P2P. Le modèle de pair-à-pair commence alors à prendre de plus en plus d'ampleur et à couvrir un spectre applicatif très large, allant des applications de partage de fichiers aux applications de calcul scientifique en passant par les applications collaboratives ou encore les applications de streaming P2P.

3.1.4.1 Partage de fichiers

Le partage de fichiers est incontestablement l'un des domaines où la technologie P2P a eu le plus de succès. Ces applications ont connu un grand engouement avec l'apparition, en juin 1999, de Napster [SGG03] et de FreeNet [CSWH01]. L'avènement des connexions Internet à haut débit, et notamment l'ADSL, sans limite de temps, a contribué à cet essor. En effet, Napster fut le premier logiciel basé sur le concept de pair-à-pair à être utilisé à grande échelle pour le partage de fichiers musicaux, au format MP3, entre les utilisateurs connectés au système. La popularité de Napster a débouché sur de nombreuses variantes d'applications de partage de fichiers, comme Gnutella [RIF02] et eDonkey/eMule [HKF$^+$06] en 2000, KaZaA [GDS$^+$03] en 2001 et BitTorrent [QS04] en 2002.

Cette lignée d'applications a permis l'échange de fichiers entre personnes directement et sans passer par des intermédiaires, de particulier à particulier (de pair à pair), chose qui est tout à fait légale. Or, il est fréquent que des individus échangent des fichiers qui ne leur appartiennent pas, par exemple des fichiers qui sont en marque déposée ou qui répondent à un droit d'auteur, etc. Les applications P2P ne sont donc pas illégales en soi mais elles peuvent être utilisées illégalement, au même titre qu'un serveur FTP peut être employé pour distribuer des fichiers illégalement.

La première génération des applications de partage de fichiers, comme Napster, a été fondée sur une architecture pseudo pair-à-pair. Aussi, KaZaA et eDonkey mettent en œuvre une architecture pseudo pair-à-pair, avec plusieurs nœuds d'indexation. Gnutella est, par contre, fondé sur une architecture pair-à-pair pur qui le rend invulnérable à la fermeture des serveurs d'indexation. BitTorrent ainsi qu'eMule sont également basés sur une architecture pair-à-pair pur mais ils emploient des structures (DHT) facilitant la recherche des fichiers partagées.

3.1.4.2 Calcul scientifique

Si le concept de P2P est à l'heure actuelle très utilisé dans le cadre du partage de fichiers, il peut se révéler être une solution élégante pour faire travailler ensemble un nombre très important de machines interconnectées via un réseau. En effet, grâce aux communications directes qui sont possibles entre les nœuds, il est envisageable de réaliser du calcul parallèle haute performance appliqué à des problèmes divisés en unités de calcul interdépendantes. La résolution de systèmes linéaires [BT89] ou de systèmes d'équations différentielles ordinaires [HW96] est un des exemples classiques de tels problèmes.

Les applications destinées à résoudre ce genre de problèmes sont généralement conçues par des scientifiques, non obligatoirement informaticiens. Il est donc d'usage de leur fournir des environnements de développement proposant des fonctionnalités pair-à-pair, qui leur facilitent la tâche de programmation. L'outil le plus générique est sans nul doute l'environnement JXTA [Gon01], puisqu'il permet de construire tout type d'application pair-à-pair. Toutefois, il existe de nombreux environnements reposant sur les architectures P2P, dédiés au calcul scientifique comme JNGI [VNRS02], basé sur JXTA, ProActiveP2P [JCCP05] de l'INRIA ou encore JaceP2P [BCV08] du LIFC.

3.1.4.3 Applications collaboratives

Les applications collaboratives sont des programmes grâce auxquels plusieurs utilisateurs interagissent simultanément, au moins pendant une partie de la durée de vie de l'application. Les communications directes entre les nœuds dans les architectures P2P apportent une bonne adaptabilité à ces applications. Les applications collaboratives qui ont été déployées sur des architectures P2P couvrent un spectre applicatif très large, qui s'étend des moteurs de recherche comme Maay [NKS06] à la résolution de noms de domaine (DNS) [CMM02] en passant par les systèmes de stockage comme OceanStore [KBC$^+$00] ou Ivy [MMGC02]. Néanmoins, les plus répandues et les plus représentatives restent les applications temps-réel, grâce auxquelles plusieurs utilisateurs agissent en même temps sur les mêmes objets. Les exemples les plus courants de telles applications sont les jeux en ligne, où les utilisateurs interagissent pour se divertir, et les messageries instantanées, où les usagers s'échangent des messages pour tenir des conversations.

De même que pour le calcul scientifique, des environnements de développement ont été proposés afin de faciliter la programmation des applications collaboratives sur des architectures P2P. Parmi ces environnements, nous citons Groove [Net00] et Magi [Bol00].

Jeux en ligne

Dans un jeu en ligne, souvent appelé *jeu massivement multijoueur*, un très grand nombre de joueurs se connectent simultanément, via un réseau, à un *univers de jeu* afin d'y collaborer ou de s'y affronter. Dans les jeux en ligne déployés sur des architectures P2P, chaque nœud envoie l'état de son environnement, les mouvements du joueur par exemple, à tous les autres nœuds. Le jeu est mis à jour uniquement lorsque toutes les mises à jour sont reçues de tous les nœuds [Lu04, BPS06]. Cette architecture a été employée dans les premières versions du célèbre jeu DOOM.

Messagerie instantanée

Une application de messagerie instantanée permet l'échange instantané de messages textuels entre plusieurs ordinateurs interconnectés via un réseau, et plus communément via Internet. Ce moyen de communication est caractérisé par le fait que les messages s'affichent instantanément (quasiment en temps réel, les contraintes temporelles n'étant pas fortes dans ces applications) et permettent un dialogue interactif, on parle de bavardage par un clavier ou d'un *clavardage*. Les applications de messagerie instantanée basées sur le concept P2P, comme par exemple P2P Messenger [Ufa08], permettent à des utilisateurs du réseau de communiquer en direct et sans passer par l'intermédiaire d'un serveur.

3.1.4.4 Streaming audiovisuel

Les applications de streaming employées sur des architectures P2P ont pris beaucoup d'ampleur ces dernière années. Le streaming pair-à-pair peut être compris de manière informelle par le streaming de particulier à particulier. Dans ce type d'application, les pairs s'échangent des contenus multimédias en utilisant le streaming comme mode de transmission. Les contenus échangés peuvent être soit des streams vivants ou alors des streams rémanents (cf. 1.2.4).

Les systèmes de streaming P2P sont très couramment utilisés avec des streams vivants. Les pairs peuvent être producteurs des streams comme dans les applications de téléphonie en ligne ou de visioconférence telles que Skype™ [Sky08]. Aussi, les pairs peuvent agir comme des relais : un pair qui reçoit un stream qu'il a choisi, retransmet le même stream à un ou plusieurs pairs le désirant, ces derniers en font de même et ainsi de suite. Notons que le stream relayé peut être aussi bien vivant que rémanent. Le streaming P2P est actuellement très populaire pour regarder des chaînes de télévision, et notamment des matchs de foot, en direct. Plusieurs réalisations de systèmes de streaming P2P sont d'ores et déjà disponibles. Nous en citons les plus connus : Joost™ [Joo08], PPLive [PPL08], PPStream [PPS08], PeerTV [Pee08b] et Babelgum™ [Bab08]. Les performances enregistrées par ces systèmes montrent qu'il est possible de déployer un service de streaming à grande échelle à très faible coût.

Les systèmes de streaming P2P, à l'instar des systèmes de partage de fichiers P2P, sont très utilisés pour le partage de contenus multimédias rémanents. Ces systèmes utilisent la technologie du streaming comme mode de transmission. C'est en ce point que réside la différence entre les deux types de systèmes. Dans la section suivante, nous détaillons plus amplement les systèmes de streaming P2P, qui constituent le centre d'intérêt de ce chapitre.

3.2 Systèmes de streaming P2P

Dans les systèmes de streaming traditionnels, la transmission de données aux usagers incombe au serveur, et à ses proxies. Les capacités de transmission des clients ne sont donc pas exploitées et restent simplement en veille. En intégrant le concept de pair-à-pair, les systèmes de streaming P2P utilisent certaines capacités de transmission en veille du côté des clients. La participation des pairs dans le streaming permet d'utiliser l'ensemble de l'infrastructure réseau, on parle alors de *streaming communautaire* ou *collaboratif*. Selon que le rôle des pairs est de remplacer le serveur de streaming ou de simplement de l'assister, on distingue deux familles de systèmes de streaming P2P : P2P (pseudo ou pur) et hybrides. Un état de lieux de ces deux architectures est présenté dans les paragraphes suivants.

3.2.1 Architectures hybrides

Dans les architectures hybrides, il existe toujours un serveur qui prend en charge le streaming des contenus multimédias aux différents pairs. Des proxies, placés stratégiquement au travers du réseau, peuvent être employés afin de l'assister comme dans COPACC [ILL07]. Les pairs, en prenant part à la diffusion globale de contenus, contribuent à alléger considérablement la charge du serveur et de ses proxies.

La participation des pairs dans le streaming est mise en œuvre par un mécanisme de *multicast applicatif*. Un serveur diffuse en continu vers un client (pair), qui à son tour, fait suivre le flux vers un autre pair qui en fait de même et ainsi de suite. En théorie, le serveur pourrait desservir des millions de personnes avec un seul flux de diffusion. Toutefois, dans la pratique, une telle approche n'est pas envisageable lorsque la chaîne des pairs acheminant successivement un contenu diffusé en continu est très longue. Ceci est dû aux contraintes temps réel inhérentes aux données audiovisuelles.

Afin d'éviter les très longues chaînes, la majorité des travaux de recherche et des logiciels propriétaires [All08], universitaires [hCRZ02] et à sources ouvertes [Pee08a], hiérarchise les pairs en utilisant des topologies *arborescentes* (« tree-based »), voir Figure 3.5. L'efficacité de retransmission dans un arbre est tributaire de deux facteurs corrélés : la profondeur et le degré de l'arbre. Augmenter la profondeur d'un arbre induit un délai plus important lors de transmission depuis la racine vers une feuille de rang maximal. Afin de raccourcir ce délai, la profondeur de l'arbre doit être minimum. Le délai peut aussi être dû aux nœuds qui constituent des goulots d'étranglement. En effet, augmenter le degré de l'arbre implique l'existence d'un nœud dont le nombre de fils est très important, ce nœud constitue par conséquent un goulot d'étranglement. Le pire des cas a lieu si l'arbre est une étoile centrée autour de la racine, alors que le nombre de goulots d'étranglement se trouve minimisé dans le cas où l'arbre est une chaîne. Cependant, les feuilles dans une chaîne subissent des délais trop importants dûs au nombre de relais. Il est donc souhaitable de réduire la profondeur de l'arbre tout en gardant son degré borné. Plusieurs systèmes ont proposé des structures garantissant un compromis entre ces deux paramètres, comme Chaining [SHT97], Overcast [JGJ+00], ESM [hCRZ00], ZigZag [THD03] et ACTIVE [ZL05].

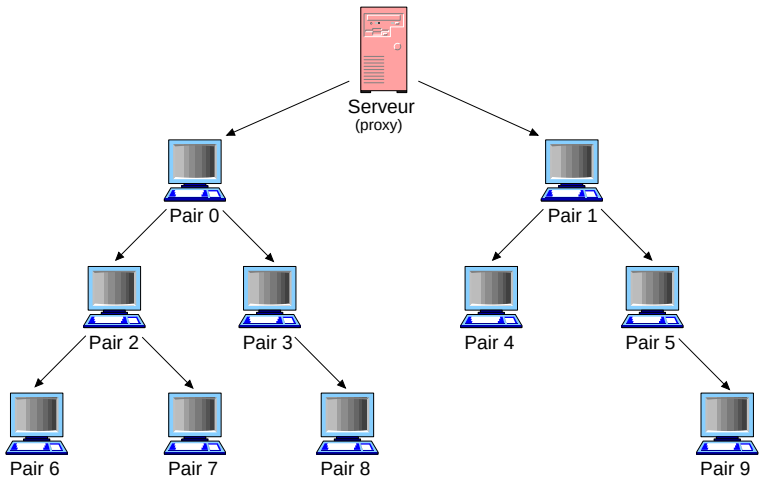
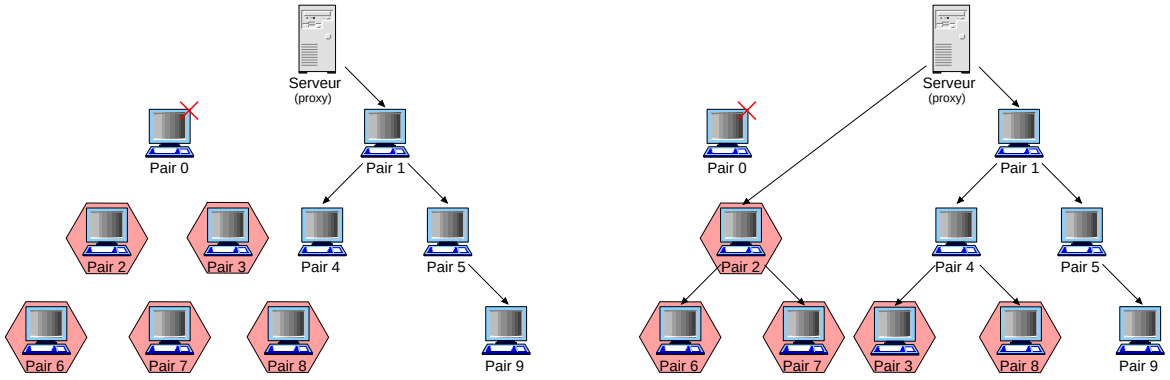


Figure 3.5 – Topologie arborescente d'un système de streaming P2P hybride avec dix pairs hiérarchisés en trois niveaux.

Les pairs ne sont pas présumés être connectés en permanence, c'est-à-dire qu'un utilisateur peut quitter le système à tout moment ; il peut éteindre son ordinateur ou se déconnecter du système, ou son ordinateur peut tomber en panne. Quel que soit le cas de figure, les pairs auxquels il faisait suivre un stream, ainsi que ceux vers lesquels ces derniers réacheminaient le flux en question, le perdront. Il contacteront alors le serveur qui devra réorganiser la pyramide (voir la Figure 3.6). La transmission serait donc interrompue jusqu'à ce qu'un autre pair soit disponible pour prendre le relais. Le temps de réparation de l'arbre peut devenir long et très décourageant dans des environnements excessivement volatils. Néanmoins, la mise en place d'un cache de lecture (cf. Section 1.4.1) peut masquer partiellement le temps de reconstruction de l'arbre [AGG+07].



(a) Impact d'une déconnexion sur l'arborescence

(b) L'arborescence après reconstruction

Figure 3.6 – Reconstruction d'un arbre de diffusion suite au départ du Pair 0.

Dans le cas des applications MOD, les techniques de batching (cf. Section 1.3.3.3) et patching (cf. Section 1.4.2.2) peuvent être employées afin d'optimiser la consommation de ressources associées à un arbre de diffusion d'un contenu, en groupant les pairs dans des sessions en fonction de leur temps d'arrivée dans le système. La figure 3.7 illustre le cas où les pairs sont « *batchés* » à un intervalle de 10 secondes. On y voit deux sessions démarrant à partir des instants 20,0 et 31,0 secondes. Notons que les pairs qui appartiennent à des sessions différentes n'interagissent pas.

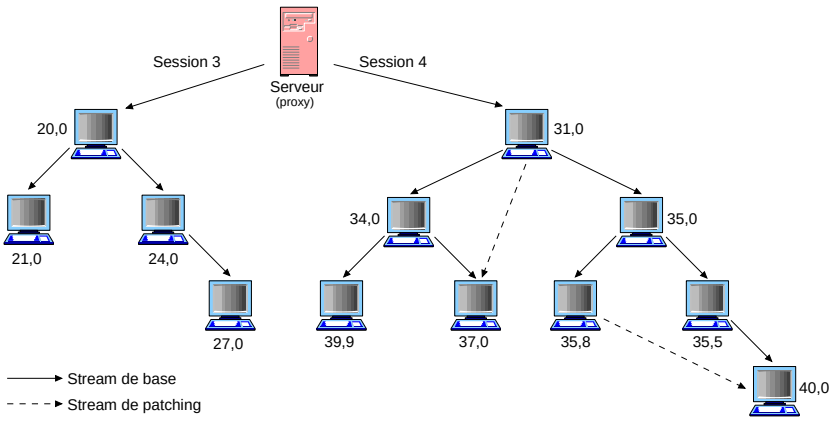


Figure 3.7 – Une partie de l'arborescence d'un système de streaming P2P hybride dans une application MOD, avec les instants de connexion des pairs.

La configuration en un seul arbre de diffusion a donné de bons résultats pour le streaming à petite échelle, avec des groupes de 10 à 100 récepteurs simultanés [SGMZ04]. Toutefois, cette configuration est limitée par la bande passante de sortie offerte par les pairs intermédiaires, d'une part, et par le fait que les nœuds qui constituent les feuilles de l'arbre ne participent pas à la retransmission des contenus, d'autre part. Ces nœuds étant majoritaires dans l'arbre de diffusion, la capacité de retransmission des pairs dans le système se trouve donc sous-exploitée.

Afin de remédier à ces problèmes, des systèmes comme SplitStream [CDK+03], CoopNet [PWCS02] et Bullet [KRAV03], encodent les contenus par descriptions multiples (cf. Section 1.1.2.2). Ils construisent ensuite plusieurs arbres de diffusion : un arbre pour chaque description d'un contenu. Un pair peut être, à la fois, une feuille dans un arbre et un nœud intermédiaire dans au moins un autre, comme le montre la Figure 3.8. Ainsi, la capacité de transmission des pairs est mieux exploitée, puisque un pair serait vraisemblablement un nœud intermédiaire dans plusieurs arbres. Dans la pratique, le nombre d'arbre dans lesquels un pair est placé en tant que nœud intermédiaire est fixé proportionnellement à sa bande passante de sortie.

Cette structuration en plusieurs arbres de diffusion rend le système plus tolérant aux pannes. La déconnexion d'un pair implique la perte d'une seule description, ce qui se traduit par une perte de la qualité du contenu diffusé. Néanmoins, le streaming du contenu en question n'est pas interrompu. La description manquante est reçue à nouveau après la réorganisation des pyramides.

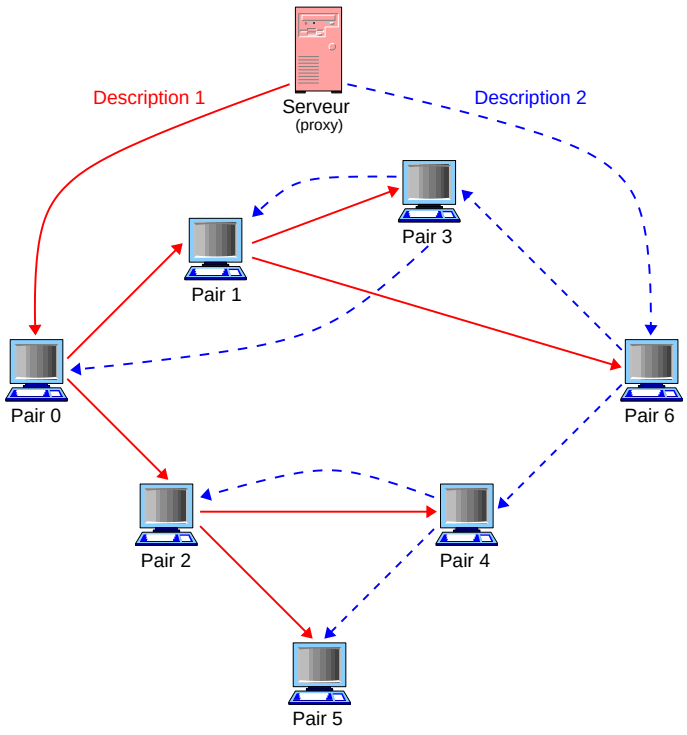


Figure 3.8 – Un système de streaming P2P hybride avec deux arbres de diffusion, chacun correspondant à une description du même contenu.

3.2.2 Architecture pair-à-pair

Dans la deuxième famille de systèmes de streaming P2P, les pairs remplacent totalement le serveur de streaming et ses proxies. Le service de streaming est assuré par les pairs pour les pairs. Les fonctionnalités de recherche de ressources peuvent également être prises en charge par les pairs (P2P pur), ou alors par un annuaire dédié mis en place à cette fin (pseudo P2P).

La disponibilité de plusieurs pairs, capables d'acheminer des données via différents chemins, est exploitée davantage dans les systèmes adoptant une architecture P2P, par rapport aux architectures hybrides avec plusieurs arbres de diffusion par contenu. En effet, les architectures P2P tirent profit de la multiplicité des chemins dans le réseau P2P en employant une *topologie maillée* (« mesh-based »). Une telle topologie permet de répondre aux scénarios où les topologies arborescentes atteignent leurs limites [MRG07]. D'ailleurs, les architectures maillées sont beaucoup plus tolérantes aux pannes que les architectures arborescentes.

En employant une topologie maillée, il est possible que de nombreux pairs collaborent pour effectuer une transmission à l'intention d'un seul récepteur. La réception simultanée de plusieurs pairs permet aux pairs récepteurs d'exploiter au maximum leurs capacités de réception, et améliore donc l'efficacité du système. Si un émetteur éteint son ordinateur ou rencontre un problème d'encombrement, conséquence de la popularité de ses contenus, le récepteur pourra toujours être desservi par d'autres émetteurs. De cette manière, personne n'est tributaire du comportement d'un seul émetteur, comme l'illustre la figure 3.9.

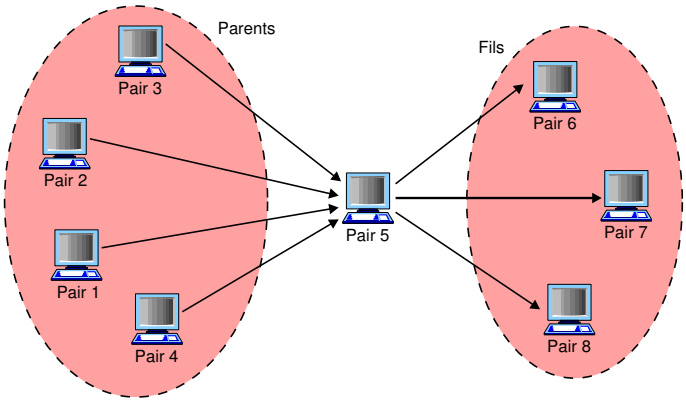


Figure 3.9 – Topologie maillée dans un système de streaming P2P.

De même, un pair peut participer à la fourniture de données à plusieurs pairs. Les contenus multimédias sont alors soit répliqués soit répartis entre plusieurs pairs. Notons que les relations émetteurs/récepteurs sont maintenues de manière dynamique dans les architectures P2P. Une multitude de systèmes adoptant cette architecture a été proposée ces dernières années comme CoolStreaming [ZLLY05], Narada [CRSZ01], SCRIBE [CDKR02], PULSE [PPKB07], PROMISE [HHB+03], GnuStream [JDXB03], Octoshape [AR03] et BiToS [VIF06].

Dans la littérature, on trouve principalement deux approches permettant de gérer la réception de données depuis plusieurs sources simultanément [3]. L'objectif recherché dans les deux cas est de garantir une restitution sans saccade (en respectant leurs contraintes temps réel intrinsèques)

3. La réception est généralement pilotée par le récepteur, c'est-à-dire le mode de transmission conforme au modèle « client pull » (cf. Section 1.3.4).

des contenus multimédias en permettant une réception simultanée de parties distinctes du même contenu via des chemin différents. Les deux approches se distinguent en fonction du choix de ces parties. Il s'agit dans la première approche de segmenter le flux en blocs de données, de façon à rapatrier des blocs différents depuis divers pairs, comme l'illustre le schéma 3.10. La deuxième méthode, plus élaborée, consiste à utiliser des contenus encodés par descriptions multiples (cf. Section 1.1.2.2.) de telle sorte que chaque description soit transmise par un pair différent.

En l'absence de serveur dédié, les systèmes de streaming P2P permettent de déployer un service de streaming à faible coût. Cependant leur instabilité soulève de nombreuses questions, notamment concernant la disponibilité des données au sein de ces systèmes. En effet, la volatilité des pairs met en cause la disponibilité des données partagées ou relayées par ces derniers. Derrière cette problématique, qui reste ouverte à ce jour, se cache un des verrous empêchant la vulgarisation des systèmes de streaming P2P. Dans la suite de ce chapitre, nous présentons notre contribution pour lever ce verrou.

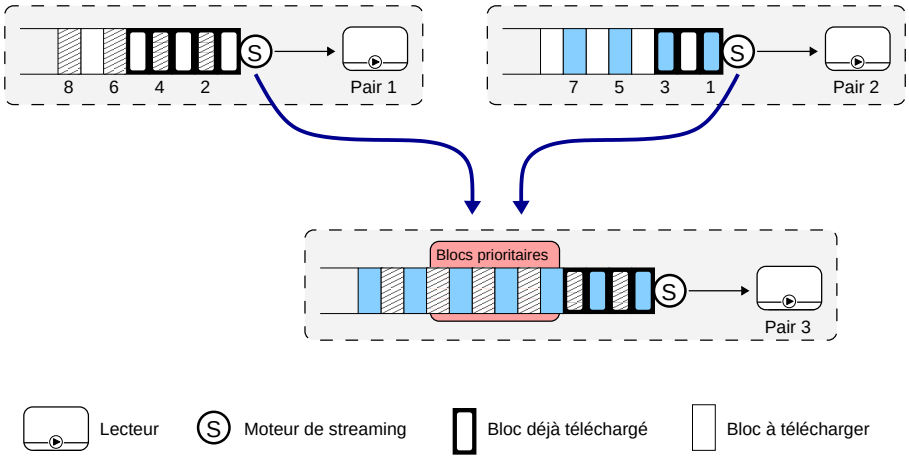


Figure 3.10 – Collaboration entre deux pairs dans la transmission de blocs de données à un troisième.

3.3 Disponibilité de données dans les systèmes de streaming P2P

Dans le reste de ce chapitre, nous nous focalisons sur les systèmes de streaming P2P. En particulier, nous nous intéressons à des systèmes à petite échelle dans lesquels un pair prend en charge le streaming de ses contenus partagés. Il y a donc un seul pair émetteur par contenu multimédia, à la différence des systèmes à grande échelle où il est tout à fait envisageable qu'un contenu soit partagé par plusieurs pairs, tous prêts à consacrer une partie de leurs ressources pour le streaming de ce contenu.

Les systèmes de streaming P2P à petite échelle peuvent être déployés sur des réseaux locaux permettant aux pairs d'avoir une large bande passante de sortie ; celle-ci est nécessaire pour que les pairs puissent prendre en charge le streaming de leurs contenus partagés. À titre d'exemple, de tels systèmes peuvent être utilisés par les étudiants au sein d'un campus pour partager leurs clips préférés, des vidéos amateurs, etc. Un scénario similaire peut se produire dans une cafétéria moderne où la clientèle partagerait des vidéos d'actualité, des sketches, des bêtisiers, etc.

La volatilité des pairs dans de tels environnements spontanés peut compromettre la disponibilité des contenus partagés par ces derniers. En effet, un pair peut quitter le système à tout instant, en particulier durant le streaming de l'un de ces contenus partagés. Ainsi, les contenus partagés par ce pair peuvent devenir indisponibles. La disponibilité des contenus partagés, et en particulier ceux auxquels on accède, est primordiale dans de tels systèmes.

La disponibilité de données est une problématique qui a attiré l'intérêt des chercheurs et des industriels au cours des dernières années. De manière générale, elle est obtenue en répliquant les données sensibles. Dans un contexte client-serveur traditionnel, plusieurs solutions ont été élaborées pour assurer la disponibilité des données jusqu'à 99,999 %. Parmi les solutions les plus populaires, nous citons les systèmes de stockage contrôlés en tant que matrice redondante de disques indépendants (RAID, cf. Section 1.3.3.5) et les systèmes de stockage en réseau (NAS). D'autres solutions visent à répliquer les données à échelle globale, comme les réseaux de diffusion de contenus (CDN) tels que Akamai [aka] ou les serveurs distribués comme OceanStore [KBC+00].

Dans le cadre de systèmes pair-à-pair, où il n'existe pas de serveur pour y stocker l'intégralité des données, le problème de la disponibilité des données est davantage accentué. En raison du départ imprévisible des pairs, atteindre une disponibilité des données à 99,999 % devient un objectif trop ambitieux. Par exemple, la persistance des données n'est pas garantie dans des systèmes de partage tels que Gnutella ou FreeNet. Néanmoins, de nombreux prototypes utilitaires comme CFS [DKK+01], PAST [RD01b] et Farsite [BDET00] se sont fixés comme but de garantir la disponibilité des données dans un système P2P. Ces prototypes militent en faveur du stockage en plusieurs copies du même fichier chez différents pairs. Cette réplication induit un trafic redondant très important au sein du système !

Nous présentons, dans cette section, notre solution pour remédier au problème de disponibilités de données dans les systèmes de streaming P2P. Nous avons choisi de consacrer les ressources du système pour garantir, dans la mesure du possible, la disponibilité de contenus visionnés par des utilisateurs qui vont vraisemblablement terminer leur visionnage. Ceci réduit le trafic introduit par la réplication, non indispensable voire inutile, de tout contenu partagé dans le système.

3.3.1 Description du problème

Avant de se pencher sur les détails de notre approche, nous établissons une description formelle du problème de la disponibilité des données dans les systèmes de streaming pair-à-pair. Pour cela, nous commençons par une illustration du problème au travers d'un scénario réel.

3.3.1.1 Exemple illustratif

Supposons que dans un lieu publique, un aéroport par exemple, les voyageurs en attente de leur vol se connectent à un système de streaming P2P afin de partager leurs contenus multimédias. Chaque utilisateur peut ainsi visionner les contenus partagés par l'ensemble des voyageurs connectés, mais aussi diffuser ses propres contenus. Les voyageurs se connectent au système par le biais des terminaux munis d'un logiciel leur permettant d'être à la fois client et hôte, un pair. Un système de streaming P2P avec quatre pairs est illustré par la figure 3.11.

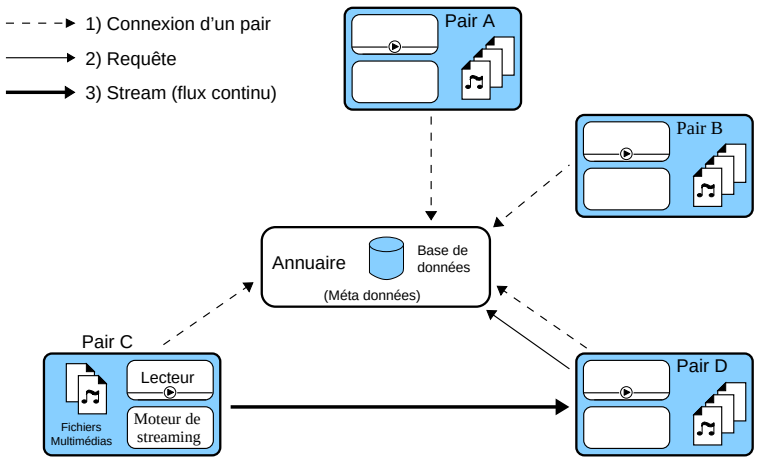


Figure 3.11 – Vue générale d'un système de streaming P2P. Les pairs A, B, C et D se connectent au système. Le pair D recherche une vidéo à regarder à travers le système (requête). Le pair C prend en charge le streaming de la vidéo demandée.

Les pairs dans un tel système permettent d'accéder à un contenu partagé de manière directe et en mode streaming. En revanche, ils ne sont pas supposés présenter un comportement de serveur, c'est-à-dire de prendre en charge un grand nombre de streams simultanément. Comme le montre la même figure, chaque pair est constitué principalement de deux composants capables de fonctionner en même temps : un lecteur multimédia et un moteur de streaming. Lorsque nous faisons référence aux fonctionnalités de son lecteur ou de son moteur de streaming, un pair est appelé respectivement, *pair client* ou *pair hôte*.

Dans le système étudié, il n'existe pas de serveur. Néanmoins, un annuaire peut être mis en place afin d'indexer les pairs connectés ainsi que les contenus rendus disponibles par ces derniers (système pseudo P2P). À son arrivée, un pair déclare à l'annuaire les contenus qu'il partage ainsi que ses capacités de streaming. De même, lorsqu'un utilisateur souhaite visionner un contenu à travers le système, le pair sous-jacent interroge l'annuaire afin d'obtenir la liste des contenus disponibles dans le système à ce moment là. Suite à une sélection de la part d'un utilisateur, le pair client envoie une *requête* à l'annuaire en précisant le contenu choisi. L'annuaire se charge de trouver un pair hôte partageant le contenu sélectionné. Dans le cas où la recherche aboutit, la requête est *acceptée* et l'annuaire informe le pair client de l'adresse IP du pair hôte élu afin que les deux pairs puissent être en communication directe. Sinon, la requête est *rejetée*.

Les systèmes de streaming P2P permettent aux utilisateurs de visionner des contenus multimédias partagés dès que les données arrivent à leurs machines, c'est-à-dire en employant le mode streaming pour la transmission de données. Néanmoins, la volatilité des utilisateurs, si caractéristique de ce genre d'applications, soulève un problème crucial qui est la disponibilité des données au sein du système. En effet, un éventuel départ d'un pair hôte mettrait en péril la disponibilité des contenus qu'il partageait ; et pire encore, les streams pris en charge par ce dernier se trouveraient interrompus. L'arrêt brutal d'un service de streaming peut être très décevant, notamment pour les utilisateurs qui sont sur le point de terminer le visionnage d'un contenu multimédia. À titre d'exemple, un utilisateur en cours de visionnage d'un épisode de la série « Dr House » sera certainement frustré si l'hôte se déconnecte à 10 minutes de la fin de l'épisode, car il ne connaîtra pas la clé de l'énigme, qui se dévoile typiquement dans les 5 dernières minutes de chaque épisode.

3.3.1.2 Formalisation du problème

Soit $k = k(t)$ le nombre d'utilisateurs connectés au système à un instant t. L'ensemble des utilisateurs connectés à cet instant peut être exprimé par $U = \{U_1, U_2, \cdots, U_k\}$. Nous supposons que chaque utilisateur (U_i ; $i \in \{1, 2, \cdots, k\}$) partage n_i vidéos à taux d'échantillonnage fixe, c'est-à-dire des vidéos de type CBR (cf. Section 1.1). Une vidéo V_l^i est caractérisée par son débit r_l^i bps et sa durée L_l^i secondes ; sa taille peut être calculée par : $r_l^i \times L_l^i$ bits. On suppose que toutes les vidéos ont un débit inférieur ou égal à un débit maximal autorisé R. L'ensemble des vidéos partagées par un utilisateur U_i est exprimé par : $V^i = \{V_1^i, V_2^i, \cdots, V_{n_i}^i\}$ où $i \in \{1, 2, \cdots, k\}$. L'ensemble des vidéos disponibles dans le système à un instant t est $V = \bigcup_{i=1}^{k} V^i$. Le nombre total des vidéos dans le système à un instant t est $|V| = n$.

Définition. Soit **regarde** une relation entre les utilisateurs connectés au système et les vidéos qui sont en train d'être regardées *via* le système par ces utilisateurs. L'existence d'un utilisateur (pair client) regardant une vidéo à travers le système implique qu'il y ait un autre utilisateur (pair hôte) prenant en charge le streaming de la vidéo en question. De manière formelle, cette relation est exprimée comme suit :

$$\boldsymbol{regarde} = \{(U_i, V_l^j) : U_i, U_j \in U; i \neq j; \exists\ l : U_i \text{ est en train de regarder } V_l^j\}$$

Soit $\overline{V}^{(t)}$ l'ensemble des vidéos qui sont en train d'être regardées à travers le système à un instant t, alors $\overline{V}^{(t)}$ peut être exprimée par :

$$\overline{V}^{(t)} = \{v : \exists\ i : (U_i, v) \in regarde\}$$

Définition. Soit **termine** une fonction sur cette relation décrivant comment le visionnage d'une vidéo s'est terminé :

$$\boldsymbol{termine} : regarde \longrightarrow \{0, 1, 2\}$$

$$(U_i, V_l^j) \longmapsto termine\left((U_i, V_l^j)\right) = \begin{cases} 2, & U_i \quad \text{annule sa requête pour visionner} \quad V_l^j \\ 1, & U_i \quad \text{termine} \quad V_l^j \\ 0, & U_i \quad \text{ne parvient pas à terminer} \quad V_l^j \end{cases}$$

La fonction *termine* a une valeur de 1 lorsqu'un utilisateur termine le visionnage d'une vidéo donnée. Cette valeur symbolise le fait que le système a réussi à transmettre la totalité de la vidéo sélectionnée au pair client. Si l'utilisateur n'a pas terminé le visionnage d'une vidéo donnée, la fonction peut avoir une valeur de 0 ou de 2. Dans le cas où l'utilisateur annule sa requête (c'est-à-dire qu'il ne veut plus regarder la vidéo) ou en cas de panne de sa connexion réseau, la valeur 2 est attribuée à la fonction termine. Nous considérons tous les autres cas où un utilisateur n'arrive pas à terminer le visionnage de la vidéo comme un échec du système, dû à l'indisponibilité de données sources. Les échecs systèmes peuvent se produire si le pair hôte quitte le système ou si sa connexion réseau tombe en panne. Ces cas d'échec sont exprimés par :

$$\boldsymbol{échec} = \{\ (U_i, V_l^j) : (U_i, V_l^j) \in regarde\ ;\ termine((U_i, V_l^j)) = 0\ \}.$$

Assurer la disponibilité des données dans le système peut être traduit par garantir la livraison de chaque vidéo v dans $\overline{V}^{(t)}$, et ce même après le départ du pair hôte de v. En d'autres termes,

garantir la disponibilité de données dans le système revient à dire qu'un utilisateur U_i regardant une vidéo V_l^j partagée par un autre utilisateur U_j ($i \neq j$) doit être capable de terminer son visionnage (s'il le souhaite), et ce même si l'utilisateur U_j se déconnecte du système. De manière formelle, cela implique qu'il n'y ait pas d'échecs dans le système :

$$|\boldsymbol{échec}| = 0.$$

Afin de garantir cette « parfaite » disponibilité des données, un cache devrait être installé pour y stocker une copie de toute vidéo dans $\overline{V}^{(t)}$, à tout instant. Or, installer un tel cache, garantissant la disponibilité des données de telle sorte que $|échec| = 0$, est impossible. En effet, compte tenu de la volatilité des pairs, il est impossible d'assurer la duplication de la totalité des données partagées par un pair avant que celui-ci ne décide de se déconnecter. Il est alors impossible, dans un environnement si volatil, de garantir la disponibilité de toutes les données partagées par tous les pairs. Le problème de disponibilité de données se réduit donc à trouver la meilleure disponibilité des données possible pour les vidéos qui sont en train d'être visionnées, plutôt que de chercher la disponibilité absolue de ces vidéos. Ainsi, la solution du problème de disponibilité de données est de trouver la configuration du cache qui minimise le nombre d'échec dans le système :

$$\min\Big(\ |\boldsymbol{échec}|\ \Big).$$

Néanmoins, cette configuration doit vérifier la condition suivante : la taille de la totalité des données cachées (mises en cache) ne doit en aucun cas dépasser la taille de l'espace de stockage du cache. Soit v_c la partie cachée d'une vidéo $v \in \overline{V}^{(t)}$ (dont le débit est r), l_c la longueur agrégée de cette partie cachée, $\overline{V}_c^{(t)}$ l'ensemble de toutes les parties cachées des vidéos qui sont en train d'être visionnées, C la taille de l'espace de stockage dans le cache ; alors cette condition peut être formulée comme suit :

$$\sum_{v_c \in \overline{V}_c^{(t)}} r \times l_c \quad \leq C \ \ ; \ \ \forall\ t \geq 0.$$

La mise en œuvre d'une telle configuration de cache soulève plusieurs interrogations liées à la fois aux caractéristiques du cache (centralisé, distribué, mixte ; taille ; etc.) et aux données (i.e. choix des suffixes à mettre en cache, leur taille, etc.). Tous ces paramètres sont dépendants les uns des autres. La taille du cache conditionne non seulement les choix et les tailles des suffixes à stocker, mais en plus elle influence les traitements (streaming) associés. En effet, dans notre cas, il ne s'agit pas de gérer un cache classique dédié uniquement au stockage des données, il faudrait en plus prendre en charge les tâches de streaming qui leur sont associées. D'autant plus, le stockage des suffixes dans le cache implique directement une consommation de la bande passante réseau. Ainsi, on fait face à un problème qui fait intervenir plusieurs paramètres entremêlés et complexes.

3.3.2 Mise en cache des suffixes

Nous développons dans cette section notre approche conçue pour améliorer la disponibilité des données dans les systèmes de streaming P2P. Elle est basée sur la mise en cache temporaire des dernières parties, que nous appelons *suffixes*, des vidéos en cours d'accès, et qui vont vraisemblablement être visionnées jusqu'à la fin par un utilisateur.

Afin de décrire le modèle général de notre approche, appelée *mise cache des suffixes* ou *suffix caching*, nous partons d'un scénario simple où le cache est centralisé. Dans la réalité, un tel cache peut être mis à disposition par un fournisseur de services. Nous définissons un modèle probabiliste pour donner la probabilité qu'un utilisateur, qui regarde une vidéo (via le système) depuis son début jusqu'à une position donnée, continue son visionnage jusqu'à la fin de la vidéo. Nous nous appuyons sur ce modèle pour décider s'il est judicieux de consacrer des ressources du système P2P afin de garantir la disponibilité d'une vidéo en cours d'accès. Dans le cas où la mise en cache d'un suffixe est jugé pertinente, ce modèle est aussi utilisé pour calculer la taille du suffixe à mettre en cache ainsi que l'instant où le processus de mise en cache sera éventuellement déclenché.

Par la suite, nous étendons notre approche vers une architecture distribuée du cache, c'est-à-dire que chaque pair fournit un petit espace cache de telle sorte que l'agrégation de ces espaces forme un *cache virtuel distribué* (DVC). Nous présentons notre politique d'allocation qui distribue les suffixes entre les pairs de manière à ce que la charge de streaming reste équilibrée entre les pairs. Ensuite, nous examinons l'impact de la variation de la taille du DVC qui est causée par le départ imprévisible des pairs. Nous décrivons notre nouvelle stratégie de gestion de cache qui est élaborée spécialement pour adapter la charge de mise en cache des suffixes de manière dynamique. Cette stratégie est fondée sur un système asservi dont le rôle est d'ajuster le nombre et la taille des suffixes à cacher en fonction de deux paramètres : la taille variable du DVC et la charge de streaming dans le système P2P.

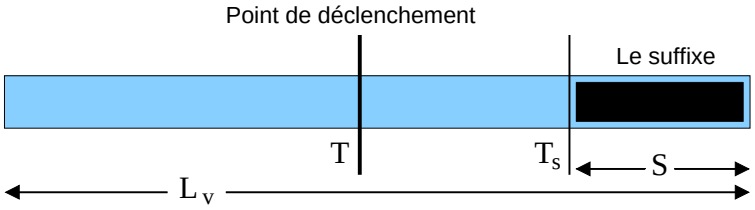


Figure 3.12 – Suffixe d'une vidéo.

3.3.2.1 Configuration centralisée

Supposons qu'un utilisateur U_i est en train de regarder une vidéo V_l^j qui est partagée par un autre utilisateur U_j. Une fois que l'utilisateur U_i atteint une position prédéfinie, appelée *point de déclenchement* (T), dans la lecture de la vidéo en question, le pair hôte (U_j) considère que l'utilisateur U_i (pair client) terminera probablement la vidéo, c'est-à-dire qu'il continuera vraisemblablement à visionner la vidéo jusqu'à la fin. Le pair hôte considérera par conséquent que la dernière partie de la vidéo V_l^j, que nous appelons *suffixe*, notée $S(V_l^j)$, est candidate à la mise en cache (voir Figure 3.12). Le pair hôte (U_j) lancera alors une *requête de suffixe* au cache, afin d'y sauvegarder temporairement le suffixe de la vidéo V_l^j.

Une requête de suffixe est dite *acceptée* si le cache accepte de commencer la sauvegarde du suffixe en question. Une requête de suffixe acceptée est dite *satisfaite* si le suffixe est intégralement copié dans le cache. Ceci sous-entend que le pair hôte est resté connecté suffisamment longtemps pour que la transmission du suffixe soit faite. Si une requête de suffixe est satisfaite, le cache sera en mesure de transmettre le suffixe au pair client, au cas où le pair hôte se déconnecterait du système après que la copie du suffixe soit terminée. La disponibilité du suffixe sera donc assurée à partir de l'instant du début du suffixe, ou simplement à partir du *début du suffixe* en considération : $T_s \ \ (T < T_s < L_l^j)$. La figure 3.13 schématise cette chaîne d'événements.

De cette manière, si un utilisateur annule une requête pour regarder une vidéo donnée avant que la lecture n'atteigne le point de déclenchement, il n'y aura pas de gaspillage des ressources du système P2P. Ceci est simplement dû au fait que le système n'a rien à faire pour garantir la disponibilité de cette vidéo. D'un autre côté, si un pair client atteint une position assez avancée dans la restitution d'une vidéo, le pair hôte peut considérer que le pair client (l'utilisateur en réalité) continuera à la regarder jusqu'à la fin, et lance par conséquent une requête de suffixe. Après la satisfaction de cette requête, il est probable que le pair hôte quitte le système au moment où le pair client est en train d'accéder au suffixe. Le cas échéant, un échec du système sera évité.

Une description concrète de notre approche passe par la réponse aux questions suivantes : comment définir l'instant (point de déclenchement) où le suffixe d'une vidéo sera candidat à la mise en cache ? Quelle est la taille de ce suffixe ? Nous déterminons la taille du suffixe, en supposant que le point de déclenchement T est connu. Ensuite, nous présentons notre modèle probabiliste utilisé pour calculer la position de ce point de déclenchement dans une vidéo. Enfin, nous examinons comment le cache est géré. Nous présentons différentes politiques identifiant le suffixe à cacher en priorité au cas où la taille du cache est limitée.

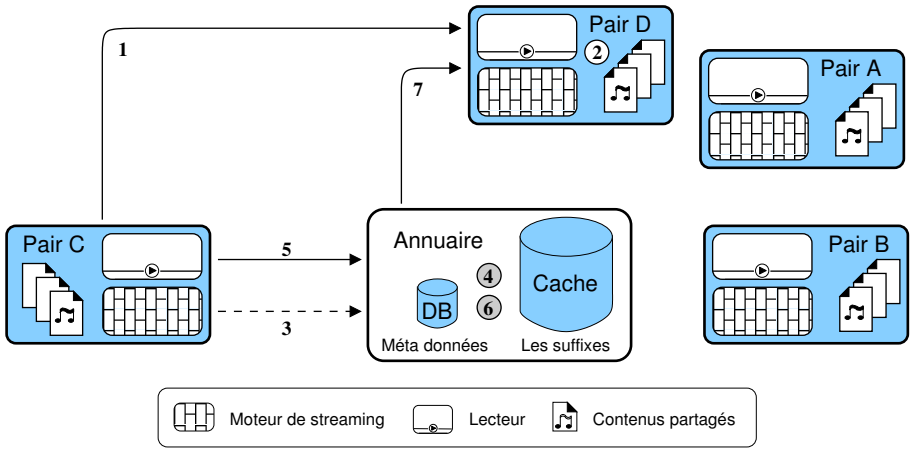


Figure 3.13 – **Mise en cache des suffixes** : 1) Le Pair C (hôte) transmet une vidéo à un Pair D (client) en streaming ; 2) Le Pair D atteint le *point de déclenchement* T ; 3) Le Pair C lance une requête de suffixe au cache ; 4) La requête de suffixe est *acceptée* par le cache ; 5) Le Pair C commence à sauvegarder le suffixe de la vidéo dans le cache ; 6) La copie du suffixe est terminée et la requête de suffixe est maintenant *satisfaite* ; 7) Le cache est en mesure de transmettre le suffixe au Pair D, si le Pair C se déconnecte.

Taille de suffixe

Supposons que le débit de transmission d'un suffixe au cache est R, le débit maximal autorisé pour les vidéos partagées dans le système. Soit L_v et r_v, la longueur et le débit de la vidéo V. La requête du suffixe de cette vidéo sera satisfaite si le suffixe est transmis dans $(T_s - T)$ secondes, où T_s et T dénotent, respectivement, le début du suffixe et le point de déclenchement du processus de la mise en cache du suffixe en considération (voir Figure 3.12). On peut donc écrire :

$$T_s - T = \frac{(L_v - T_s) \times r_v}{R}, \text{ ce qui nous conduit à : } T_s = \frac{L_v \cdot r_v + T \cdot R}{R + r_v}$$

La taille du suffixe peut être calculé par : $[(L_v - T_s) \times r_v]$. Les données du suffixe seront donc transmises à partir du bloc dont l'indice au sein de la vidéo est donné par :

$$\left\lceil \frac{L_v \cdot r_v + T \cdot R}{R + r_v} \right\rceil$$

En supposant que toutes les vidéos dans le système aient le même débit R, le début du suffixe équivaudrait à $T_s = T + (L_v - T)/2$. La taille du suffixe résultant serait $\left\lceil \frac{(L_v - T)}{2} \right\rceil \times R$. Ceci signifie que si une requête de suffixe est acceptée et si le pair hôte reste connecté pour, au moins, la moitié du temps nécessaire pour que le pair client atteigne la fin de la vidéo, la requête de suffixe sera satisfaite et la disponibilité de la vidéo en question sera assurée.

Point de déclenchement de la mise en cache

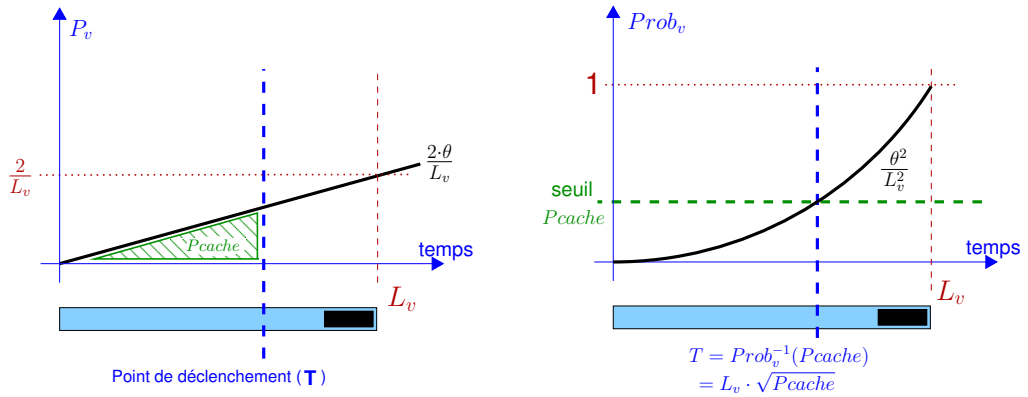
Soit la variable aléatoire continue[4] θ (secondes) représentant la position temporelle qu'un utilisateur a atteint dans la lecture de sa vidéo v via le système. Soit $Prob_v(\theta)$ la *fonction de répartition* sur cette variable donnant la probabilité qu'un utilisateur qui regardait la vidéo v depuis son début jusqu'à la position θ la termine, c'est-à-dire qu'il aille jusqu'au bout de la vidéo (L_v). La *densité de probabilité* $P_v(\theta)$, si elle existe, est la dérivée de la fonction de répartition : $P_v(\theta) = Prob_v{}'(\theta)$. Les deux fonctions sont reliées comme suit : $Prob_v(\theta) = \int_0^\theta P_v(x) \cdot dx$.

Un utilisateur qui atteint la position L_v dans la vidéo v sous-entend qu'il l'a pratiquement terminée. La probabilité qu'il termine la vidéo v est donc : $Prob_v(L_v) = \int_0^{L_v} P_v(x) \cdot dx = 1$. D'un autre côté, si un utilisateur n'a pas démarré la lecture d'une vidéo v, la probabilité qu'il la termine est nulle : $Prob_v(\theta) = \int_0^0 P_v(x) \cdot dx = 0$.

La forme exacte de la fonction de répartition $Prob_v$ dépend de l'utilisateur qui regarde la vidéo et de son contenu : long métrage, un sketch de comédien, des actualités, etc. Nous supposons que plus longtemps un utilisateur regarde une vidéo, plus forte serait la probabilité qu'il la termine. Ceci implique que la fonction de répartition est strictement croissante et par conséquent bijective.

Un suffixe d'une vidéo serait candidat à la mise en cache lorsque la probabilité, qu'un pair client termine cette vidéo, dépasse un certain seuil $Pcache$ $(0 \leq Pcache \leq 1)$. Le seuil $Pcache$ est un paramètre crucial dans notre approche et est examiné plus en détails ultérieurement. Le point de déclenchement est défini comme suit : $T = Prob_v{}^{-1}(Pcache)$.

4. Le lecteur est invité à réviser les notions de base en probabilité continue : variable aléatoire à valeurs dans $\mathbb{R}$, densité de probabilité, fonction de répartition, etc.



(a) Densité de probabilité (b) Fonction de répartition

Figure 3.14 – Fonctions de probabilité.

Nous supposons que la densité de probabilité existe et a une forme linéaire : $P_v(\theta) = a \cdot \theta$. On peut alors obtenir la fonction de répartition comme suit :

$$Prob_v(\theta) = \int_0^\theta P_v(x) \cdot dx = \int_0^\theta (a \cdot x) \cdot dx\,;$$
$$\int_0^{L_v} (a \cdot x) \cdot dx = 1 \quad \Rightarrow \quad a \cdot \left[\frac{x^2}{2}\right]_0^{L_v} = 1 \quad \Rightarrow \quad a = 2/L_v^2.$$

La fonction de répartition est donc donnée par :

$$Prob_{\boldsymbol{v}} : [0, L_v] \longrightarrow [0, 1]$$
$$\theta \longmapsto Prob_v(\theta) = \int_0^\theta P_v(x) \cdot dx = \int_0^\theta \frac{2 \cdot x}{L_v^2} \cdot dx = \frac{\theta^2}{L_v^2}$$

Le point de déclenchement, représentant un temps et étant donc positif, est ainsi calculé par :

$$Pcache = \frac{T^2}{L_v^2} \quad \Rightarrow \quad T = L_v \cdot \sqrt{Pcache} \tag{3.1}$$

Pcache	**Instant du début de suffixe**	**Taille de suffixe**
0.01	$10\% \cdot L_v$	$45\% \cdot L_v \times R$
0.03	$17\% \cdot L_v$	$41\% \cdot L_v \times R$
0.05	$22\% \cdot L_v$	$39\% \cdot L_v \times R$
0.07	$26\% \cdot L_v$	$37\% \cdot L_v \times R$
0.1	$32\% \cdot L_v$	$34\% \cdot L_v \times R$
0.2	$45\% \cdot L_v$	$27\% \cdot L_v \times R$
0.4	$63\% \cdot L_v$	$18\% \cdot L_v \times R$
0.6	$77\% \cdot L_v$	$11\% \cdot L_v \times R$
0.8	$89\% \cdot L_v$	$5\% \cdot L_v \times R$
0.9	$95\% \cdot L_v$	$2\% \cdot L_v \times R$

Tableau 3.1 – Plusieurs tailles de suffixes et leurs instants de début en fonction de ***Pcache***, calculés en utilisant l'équation (3.1).

Gestion du cache

Comme nous l'avons mentionné précédemment, le rôle du paramètre *Pcache* est crucial dans notre approche. Il contrôle la charge de la mise en cache des suffixes dans le système. En effet, cette charge est « pilotable » en faisant varier les valeurs attribuées au paramètre *Pcache*. Une faible valeur de *Pcache* génère un nombre plus important de requêtes de suffixe dont la taille est plus importante, alors qu'une grande valeur de *Pcache* a l'effet inverse.

Lorsque la valeur de *Pcache* est très faible, les pairs hôtes lancerons des requêtes de suffixe tellement tôt qu'une partie de ces requêtes concerneront des utilisateurs qui pourraient annuler rapidement leurs demandes. Afin de limiter le gaspillage des ressources du système P2P sur des requêtes de suffixe qui concernent des utilisateurs n'ayant pas avancé assez loin dans la lecture d'une vidéo, *Pcache* est minoré : $Pcache \geq Pcache_{inf} > 0$.

D'un autre côté, lorsque *Pcache* a une valeur très importante, les pairs hôtes lancerons des requêtes de suffixe tellement tard que la satisfaction de celles-ci deviendront sans importance. Ceci est dû au fait que les suffixes seront trop petits, au point qu'il sera inutile de les sauvegarder en cache. D'ailleurs, si un pair hôte est resté connecté si longtemps pour lancer une requête tardive, il est probable qu'il le restera un peu plus pour que le pair client finisse la restitution de la vidéo en question, et cela sans utilisation de cache. Afin que les requêtes de suffixe soient utiles, c'est-à-dire que les suffixes n'aient par une taille trop petite, *Pcache* est majoré : $Pcache \leq Pcache_{sup} < 1$.

Une requête de suffixe est considérée comme inutile si la taille du suffixe résultant est inférieure à 5% de la taille de la vidéo d'origine. Par ailleurs, une requête de suffixe ne doit être lancée que si l'utilisateur (pair client) a visionné au moins 10% de la vidéo correspondante. En observant la variation de la taille des suffixes en fonction de *Pcache* (cf. Tableau 3.1), nous avons choisi la configuration suivante pour les limites de *Pcache* : $Pcache_{inf} = 0.01$; $Pcache_{sup} = 0.8$.

En raison de la limitation de son espace de stockage, il est possible que le cache ne soit pas capable de satisfaire toutes les requêtes de suffixes, et ce même si la limite supérieure est attribuée à *Pcache*. Il convient donc d'accepter les requêtes de suffixe en fonction d'une *priorité* qui leur est attribuée. Ainsi, le cache accepte les requêtes de suffixe tant que son espace de stockage le permet, mais en commençant par les requêtes ayant la priorité la plus élevée. Suite à l'acceptation d'une requête, le cache réserve un espace de stockage suffisamment grand pour y préserver le suffixe en question. Cet espace ne peut être employé pour la sauvegarde d'autres suffixes uniquement si l'on se trouve dans l'un des cas suivants : l'utilisateur termine la vidéo associée au suffixe et il n'y a aucun autre utilisateur regardant la même vidéo ; l'utilisateur se rétracte sur sa demande et il n'y a aucun autre utilisateur regardant la même vidéo. L'espace réservé est donc associé à une vidéo. Nous avons testé trois métriques permettant de définir une priorité sur les requêtes de suffixes :

- **Premier arrivé premier servi** (First Come First Served, FCFS) : comme son nom l'indique, les requêtes seront traitées dans l'ordre de leur arrivée, sans optimisation aucune de l'usage de l'espace de stockage du cache.
- **Le plus petit suffixe d'abord** (Smallest Suffix First, SF) : les requêtes dont le suffixe est le plus petit sont traitées en premier. Cette métrique permet d'accepter un maximum de requêtes, cependant la popularité des vidéos n'est pas prise en considération.
- **Le suffixe le plus demandé d'abord** (Most Demanded First, MDF) : les requêtes sont triées en fonction de la popularité des vidéos. Le cache est ensuite utilisé pour stocker en premier lieu les suffixes qui correspondent aux vidéos les plus populaires.

3.3.2.2 Configuration distribuée

Nous allons maintenant étendre notre approche afin de s'accommoder du cas d'un *cache virtuel distribué*, ou *Distributed Virtual Cache* (DVC) en anglais, qui est formé par l'agrégation des espaces cache fourni par les pairs (voir Figure 3.15). Cette architecture distribuée a un plus grand potentiel qu'un cache centralisé. En effet, elle permet de s'affranchir des limitations bien connues des architectures centralisées : une panne du cache compromettra la disponibilité de l'intégralité des suffixes qui y sont sauvegardés ainsi qu'une évolutivité (passage à l'échelle) réduite en raison des restrictions de la bande passante et de l'espace de stockage, qui sont inhérentes aux architectures à point d'accès unique. Par ailleurs, notons que l'agrégation des espaces de stockage fournis par les pairs amplifie la capacité du DVC à sauvegarder des suffixes avec l'augmentation du nombre des fournisseurs au fil du temps.

Le DVC peut être considéré comme une couche permettant au lecteur multimédia d'un pair et à son moteur de streaming d'accéder à cet espace cache non contigu comme s'il s'agissait d'un cache centralisé. Or, en raison de la volatilité des pairs, la nature du DVC est non seulement distribuée mais aussi dynamique. Son espace de stockage est en effet tributaire du nombre de pairs dans le système et de leur bonne volonté à fournir une partie de leur espace de stockage. L'extension de la mise en cache de suffixe dans un espace virtuel distribué entraîne donc plusieurs questions :

1. comment garantir un bon équilibrage de la charge de streaming entre les pairs de telle sorte que, d'une part, la disponibilité des données soit maintenue, et que d'autre part, aucun pair ne soit accablé par le streaming des suffixes ?
2. comment gérer la variation de la taille du DVC, qui est causée principalement par les arrivées et les départs imprévisibles des pairs ?

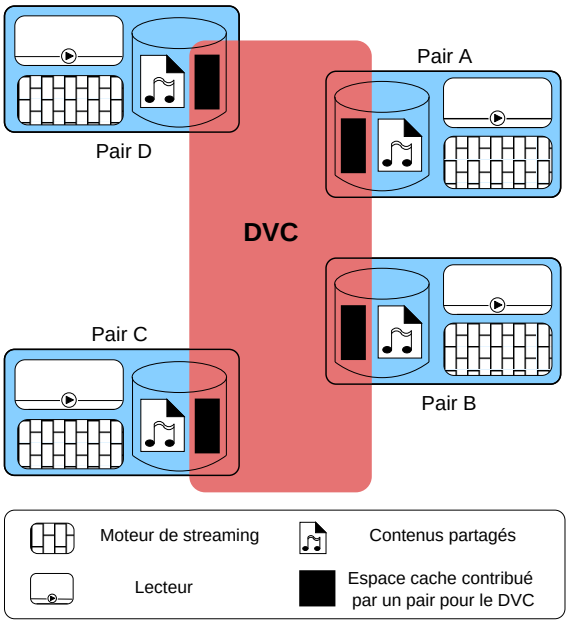


Figure 3.15 – **Cache virtuel distribué (DVC)** : formé par l'agrégation des espaces de cache fournis par les pairs.

Afin de traiter le premier problème, une politique d'allocation de suffixes a été mise au point pour distribuer « judicieusement » ces derniers entre les pairs. Ensuite, pour faire face à la seconde problématique, nous avons élaboré une stratégie adaptative dont le rôle est de gérer la mise en cache des suffixes au sein du DVC. Cette stratégie s'appuie sur un système asservi afin d'ajuster le paramètre *Pcache* de sorte que le nombre et la taille des suffixes à mettre en cache soient adaptés à l'espace de stockage disponible dans le DVC.

Politique d'allocation de suffixe

Soit C_i ($C_i > 0$) la taille de l'espace cache fourni par le pair U_i. Étant donné que le DVC est formé par l'agrégation des espaces cache contribués par les pairs, sa taille peut être exprimée par la somme : $C = \sum_{i=1}^{k} C_i$.

Afin de ne pas gaspiller l'espace de stockage du DVC, les données mises en cache n'y sont pas répliquées. Ceci a l'avantage d'éliminer la surcharge qui pourrait résulter de la réplication, comme par exemple la communication supplémentaire ou le traitement de la cohérence entre les répliques. En revanche, le fait de ne pas répliquer les suffixes augmente la sensibilité du DVC au départ des pairs. Nous avons tout de même observé des performances très satisfaisantes dans les expériences que nous avons menées, sans réplication des suffixes (cf. Section 3.4).

Dans le système de streaming P2P considéré, il ne peut y avoir qu'un seul pair émetteur par stream, c'est-à-dire nous avons écarté la possibilité que plusieurs pairs participent à la transmission d'un contenu donné. Les suffixes ne sont donc pas répartis entre les pairs. Un suffixe $S(v)$ d'une vidéo v est *intégralement* caché dans un espace cache fourni par un usager différent des deux utilisateurs : celui qui regarde la vidéo et celui qui l'héberge.

Lorsqu'un suffixe est mis en cache chez un pair, le moteur de streaming de ce dernier prend en charge la transmission de ce suffixe, si besoin est. Or, dans la pratique, les utilisateurs autorisent uniquement quelques streams à être pris en charge par leurs machines, afin qu'elles ne soient pas submergées par la charge de streaming. Typiquement, un ensemble restreint de threads de transmission est mis de côté par chaque pair pour gérer les streams. Notre politique d'allocation prend cet aspect en considération afin d'équilibrer la charge de streaming entre les pairs. Lorsqu'un suffixe doit être mis en cache, il est sauvegardé chez le pair possédant le plus de threads disponibles et dont l'espace de stockage fourni au système permet de stocker ce suffixe.

Soit X_i, $\overline{X}_i(t)$ et $\overline{C}_i(t)$, respectivement, le nombre maximum de threads autorisés par le pair U_i, le nombre de threads utilisés par ce pair à un instant t et l'espace cache occupé par des suffixes chez le même pair à l'instant t. Soit $\overline{U}(t)$ le sous-ensemble des pairs connectés et qui sont en mesure de sauvegarder dans leur espace cache le suffixe $S(V_l^j)$ d'une vidéo V_l^j hébergée par le pair U_j et en cours d'accès par le pair U_i ($i \neq j$), à un instant t :

$$\overline{U}(t) = \{U_\omega \in U : \overline{C}_\omega(t) > r_l^j \times L_l^j \ ; \ \omega \neq i \ ; \ \omega \neq j \ ; \ \overline{X}_\omega(t) < X_\omega\}$$

Le suffixe $S(V_l^j)$ serait alors sauvegardé dans l'espace cache contribué par le pair $U_\Omega \in \overline{U}$, qui possède le plus de threads disponibles, à l'instant t :

$$X_\Omega - \overline{X}_\Omega(t) = \max_{U_\omega \in \overline{U}(t)} X_\omega - \overline{X}_\omega(t)$$

Stratégie de gestion adaptative

La taille du DVC varie avec l'arrivée et le départ des utilisateurs au fil du temps. En outre, l'activité des utilisateurs connectés oscille, elle aussi, avec le temps, induisant une variation dans la charge de streaming au sein du système. Étant donné que le paramètre *Pcache* contrôle le nombre de requêtes de suffixe ainsi que la taille des suffixes générés, il est possible d'influer sur la charge de mise en cache des suffixes en modifiant la valeur attribuée à ce paramètre. L'objectif recherché par notre stratégie est d'ajuster dynamiquement la valeur de *Pcache* de manière à tirer le meilleur rendement possible du DVC, en terme de disponibilité des données.

Lorsqu'il y a peu d'utilisateurs dans le système, le cache pourrait sauvegarder des suffixes plus grands et en plus grand nombre. Dans ce cas, *Pcache* doit être diminué pour générer un nombre plus important de requêtes de suffixe, chaque suffixe ayant une taille plus grande. Autrement dit, *Pcache* doit être atténué afin de générer un volume plus important de données à mettre en cache, et par conséquent d'améliorer la disponibilité des données au sein du système. L'atténuation de *Pcache* doit être faite de sorte que l'espace de stockage du DVC soit exploité au maximum, c'est-à-dire que *Pcache* doit être configuré au plus bas possible, sans pour autant dépasser la limite inférieure $Pcache_{inf}$.

En revanche, lors d'une forte charge de streaming dans le système, il pourrait y avoir plus de données à stocker que le DVC ne peut en contenir, c'est-à-dire qu'il pourrait y avoir plus de requêtes de suffixe que le DVC ne peut en accepter. Dans ce cas là, *Pcache* doit être augmenté pour réduire le nombre de requêtes de suffixe générées ainsi que la taille des suffixes demandés. Autrement dit, *Pcache* doit s'accroître afin de générer un volume moins important de données à mettre en cache, et par conséquent de permettre au DVC de remplir sa fonction principale, à savoir assurer la disponibilité des données au mieux possible. Afin de garantir un bon fonctionnement du DVC, *Pcache* doit être configuré aussi haut que nécessaire mais sans dépasser la limite supérieure $Pcache_{sup}$.

Afin d'ajuster *Pcache* dynamiquement, nous utilisons un système asservi [5] (voir Figure 3.16). Un tel système prend en compte le rendement du DVC (la sortie) pour réguler le volume des données à cacher en calculant (*contrôleur* ou *régulateur*) une valeur convenable de *Pcache* (l'entrée).

Généralement, l'ajustement de l'entrée (*Pcache*) dans un système asservi se fait en fonction de l'écart entre le rendement mesuré (la sortie) et le rendement désiré. Nous mesurons le rendement du DVC avec *le nombres de requêtes de suffixe refusées par ce dernier*. Ce nombre varie en fonction du temps ($Ref{=}Ref(t)$) et indique la façon avec laquelle la taille du DVC répond à la charge de la mise en cache des suffixes dans le système.

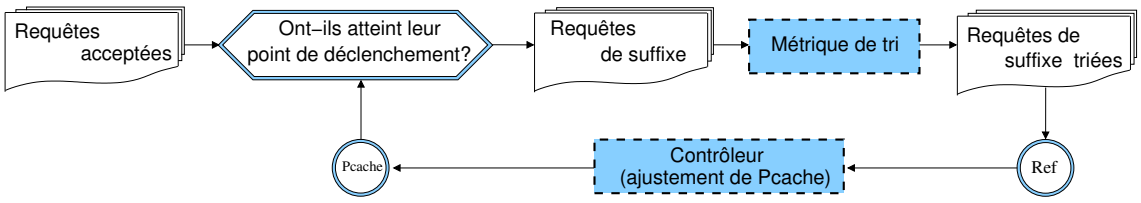


Figure 3.16 – Le système asservi, avec tri des requêtes de suffixe.

5. Un système dont la sortie est branchée sur son entrée en boucle fermée.

Un nombre nul de requêtes de suffixe refusées ($Ref = 0$) indique que le DVC est en mesure de stocker plus de suffixes de plus grande taille. $Pcache$ est donc baissé afin d'augmenter le volume des données à mettre en cache. Tant que le nombre de requêtes refusées par le DVC est nul, $Pcache$ est diminué lentement mais constamment, c'est-à-dire qu'il est décrémenté de manière linéaire par le biais d'une petite valeur ($Pcache_{step}$). La diminution se poursuit tant que $Pcache$ n'a pas atteint sa limite inférieure.

En revanche, la présence de requêtes de suffixe refusées ($Ref > 0$) indique que la charge de la mise en cache de suffixes est supérieure à celle que le DVC est capable de gérer. Dans ce cas, $Pcache$ est augmenté afin de réduire le volume des données à mettre en cache. L'augmentation de $Pcache$ continue tant que le nombre de requêtes refusées par le DVC est supérieur à zéro, jusqu'à ce que la limite supérieure de $Pcache$ soit atteinte. D'ailleurs, afin d'améliorer la réactivité du DVC à la croissance soudaine de la charge de streaming dans le système, l'augmentation de $Pcache$ est effectuée plus rapidement que sa diminution. En effet, l'augmentation de $Pcache$ est tributaire du nombre de requêtes de suffixe refusées, mais de manière limitée. L'ajustement dynamique de $Pcache$ est ébauchée par l'algorithme 2.

Algorithme 2 – Adaptation de $Pcache$

```
si (Ref = 0) alors
    si (Pcache − Pcache_step ≥ Pcache_inf) alors
        Pcache ⇐ Pcache − Pcache_step
    fin
fin
si (Ref > 0) alors
    BigStep ⇐ Pcache_step × ((Ref mod 10) + 1)
    si (Pcache + BigStep ≤ Pcache_sup) alors
        Pcache ⇐ Pcache + BigStep
    fin
fin
```

Comme le confirment les résultats de nos expérimentations, présentées dans la prochaine section, notre politique adaptative régule le volume des données à cacher en temps réel et peut par conséquent se passer du tri effectué sur les requêtes de suffixe. Le système asservi utilisé dans notre stratégie, et qui est schématisé par la Figure 3.17, induit donc une complexité algorithmique très faible. En effet, l'algorithme 2 a un temps d'exécution constant, peu importe le nombre de requêtes de suffixe dans le système. Notre approche adaptative correspond donc parfaitement au caractère à la fois très dynamique et distribué des systèmes de streaming P2P.

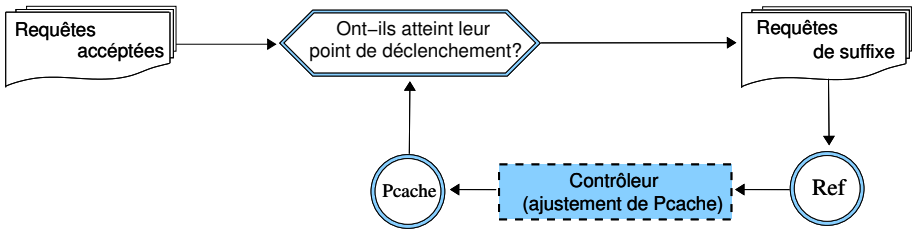


Figure 3.17 – Le système asservi, sans tri des requêtes de suffixes.

3.4 Étude de performances

Cette section présente les expérimentations que nous avons mises en œuvre afin d'évaluer les méthodes présentées précédemment. L'objectif de ces expérimentations est double. Il s'agit d'une part de justifier l'utilisation d'un cache virtuel distribué afin d'améliorer la disponibilité des données dans un système de streaming P2P. Nous nous sommes donc mis à mesurer les gains obtenus face aux coûts consentis, en terme de trafic supplémentaire dû à la mise en cache des suffixes. D'autre part, nous nous focalisons sur l'évaluation de l'efficacité de notre politique adaptative de gestion de cache. À cette fin, nous avons étudié l'impact de différents paramètres sur notre approche et ceci sous diverses configurations.

Deux métriques sont utilisées afin évaluer la pertinence des protocoles proposés. La première cherche à mesurer les gains apportés par notre approche en terme de disponibilité de données. Il s'agit du pourcentage des requêtes qui ont pu être terminées grâce à la mise en cache des suffixes, par rapport à toutes les requêtes terminées. La deuxième permet de mesurer le coût de la mise en cache des suffixes en terme de trafic supplémentaire introduit dans le système par nos protocoles, par rapport au trafic total dans le système.

3.4.1 Scénarios de tests

Les expérimentations que nous avons conduites sont effectuées à l'aide d'un simulateur spécialement développé pour analyser nos protocoles et évaluer ainsi leur efficacité. Les utilisateurs connectés au système de streaming P2P simulé sont sensés chercher des vidéos à regarder en streaming. Lorsqu'un usager souhaite regarder une vidéo via le système, il lance une requête qui peut, comme nous l'avons vu, être acceptée ou rejetée. Une requête acceptée peut avoir trois finalités : l'utilisateur annule sa demande, il termine la vidéo, ou alors il ne peut pas la regarder jusqu'à la fin suite à l'indisponibilité de celle-ci. Quel que soit le cas de figure, un utilisateur est supposé envoyer d'autres requêtes, qui peuvent à leur tour être acceptées ou rejetées. Cette procédure se poursuit tant que l'utilisateur est connecté au système.

Les paramètres du système simulé peuvent être répartis en trois catégories : ceux concernant les vidéos partagées, ceux concernant le comportement des usagers, ainsi que ceux de la configuration de leurs composants logiciels. Sauf contre-indication explicite, les valeurs par défaut que nous avons utilisées pour ces paramètres sont récapitulées dans le Tableau 3.2 et détaillées ci-après. Nous nous sommes largement inspirés de la réalité dans la construction de nos scénarios de test, que ce soit pour les caractéristiques des données, les profils des utilisateurs, ou pour les configurations matérielles.

Vidéos partagées : Le nombre des vidéos partagées par un utilisateur est une variable aléatoire, distribuée uniformément entre 7 et 15 vidéos, chacune ayant un débit de 512 Kbits/s. La durée de ces vidéos est elle aussi une variable aléatoire distribuée de manière uniforme entre 2 et 6 minutes.

Comportement des usagers : L'arrivée des utilisateurs a été modélisée par un processus Poissonnien avec une fréquence d'arrivée $\lambda = 2$, c'est-à-dire que la durée moyenne entre l'arrivée de deux usagers successifs est égale à $0,5$ secondes. Un utilisateur réside dans le système pendant une durée de séjour qui est, quant à elle, modélisée par un processus normal avec une moyenne de $\mu = 15$ minutes et un écart type $\sigma = \sqrt{2}$ minutes. Ainsi, 99% des usagers restent connectés au système pendant une durée de 9 à 21 minutes. Comme nous l'avons mentionné précédemment,

un utilisateur envoie les requêtes tant qu'il est connecté au système. Son choix de la vidéo à regarder, parmi les vidéos disponibles dans le système, a été modélisé par un processus qui suit la distribution Zipf[6] avec un facteur d'asymétrie $\theta = 0,271$. Il a été montré que cette distribution reflète bien les vidéos sélectionnées par des utilisateurs [AWY96]. Le temps mis par un utilisateur pour chercher une vidéo qui l'intéresse est modélisé par une variable aléatoire uniformément distribuée entre 0 et 2 minutes.

Configuration des pairs : Nous avons considéré que le nombre maximal de threads de transmission autorisés par un pair est une variable aléatoire distribuée de façon uniforme entre 30 et 40 threads, chacun à 512 Kbits/s. La taille de l'espace cache contribué par un pair a été modélisée par un processus normal avec une moyenne de $\mu = \alpha$ Mo et un écart type de $\sigma = \sqrt{10}$ Mo, où $\alpha \in \{50, 100, 150, 200\}$.

Catégorie	Paramètre	Valeur par défaut
Général	Durée de simulation	17 heures
Vidéos partagées	Nombre de vidéos par pair	Distribution uniforme (7 - 15)
	Durée d'une vidéo	Distribution uniforme (2 - 6) min
	Débit d'une vidéo	512 Kbits/s
Comportement des pairs	Arrivée d'un pair	Processus de poisson d'intensité ($\lambda = 2$) c.-à-d., un pair arrive toutes les $0,5$ secondes
	Durée de séjour d'un pair	Distribution normale ($\mu = 15$ min, $\sigma^2 = 2$ min) c.-à-d., dans 99% des cas cette durée $\in$ [9 ; 21] min
	Temps de recherche d'une vidéo à regarder	Distribution uniforme (0 - 2) min
	Choix de la vidéo à regarder	Distribution Zipf ($\theta = 0,271$)
Configuration des pairs	Nombre maximum de threads autorisés	Distribution uniforme (30 - 40)
	Débit de transmission sur un canal	512 Kbits/s
	Espace cache contribué par un pair	Distribution normale ($\mu = \alpha$, $\sigma^2 = 10$ Mo) $\alpha \in \{50, 100, 150, 200\}$ Mo

Tableau 3.2 – Valeurs par défaut des différents paramètres considérés dans les expérimentations.

6. Avec cette distribution, et en classant les vidéos dans le système dans l'ordre décroissant de leur popularité, la vidéo la plus populaire serait sélectionnée 8000 fois, la dixième 800 fois, la centième 80 fois et la millième 8 fois.

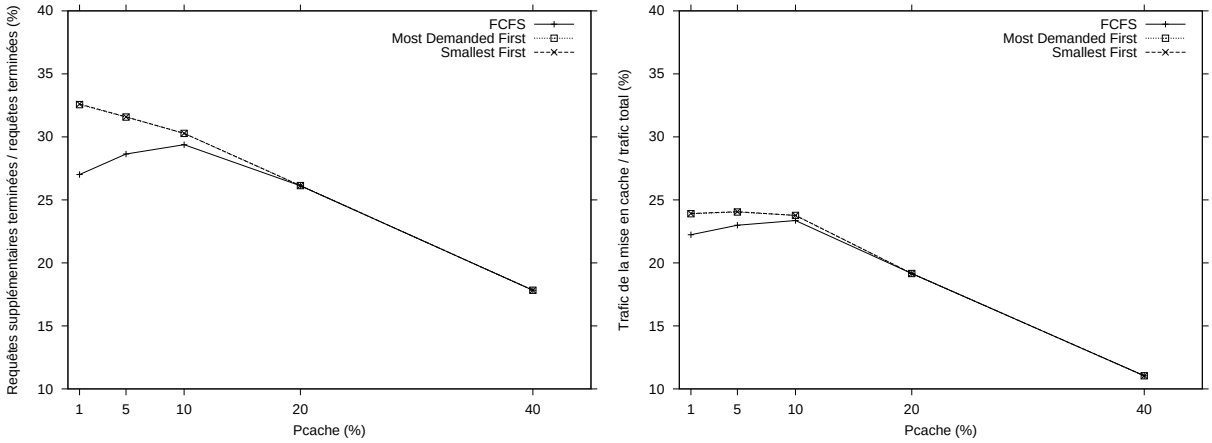
Nous avons simulé ce système pendant 17 heures, puis nous avons sauvegardé le scénario produit afin d'analyser l'impact des différents paramètres de notre approche sur la disponibilité des données au sein du système. Nous avons recensé 88258 requêtes acceptées, ce qui peut être traduit par $1,5$ requête par seconde en moyenne. Aussi, nous y avons dénombré environ 450 usagers connectés simultanément. La taille agrégée des vidéos disponibles dans le système est donc de $11 \times 4 \times 450/60 = 330$ heures de vidéos, en moyenne.

3.4.2 Résultats

Nous nous intéressons dans un premier temps aux résultats obtenus vis-à-vis du paramètre principal de notre approche, en l'occurrence *Pcache*. Dans un deuxième temps, nous analysons la performance de nos protocoles dans différentes configurations de l'espace cache fourni par les pairs. Enfin, nous examinons de plus près le rendement de notre stratégie adaptative de gestion de cache et nous concluons cette section par une discussion mettant en lumière les avantages de notre approche.

3.4.2.1 Impact du paramètre *Pcache*

Afin de mesurer l'impact du paramètre *Pcache* sur la disponibilité des données, nous avons désactivé son adaptation dynamique et fait varié sa valeur de $0,01$ à $0,4$. Nous avons utilisé une taille moyenne pour l'espace cache contribué par les pairs ($\alpha = 100$ Mo). La Figure 3.18 représente, en fonction de la valeur attribuée à *Pcache*, le gain apporté par la mise en cache des suffixes, en terme de requêtes terminées grâce à cette mise en cache, ainsi que le trafic supplémentaire qu'elle provoque.



(a) Gain (b) Coût

Figure 3.18 – Impact du paramètre *Pcache* sur la disponibilité des données.

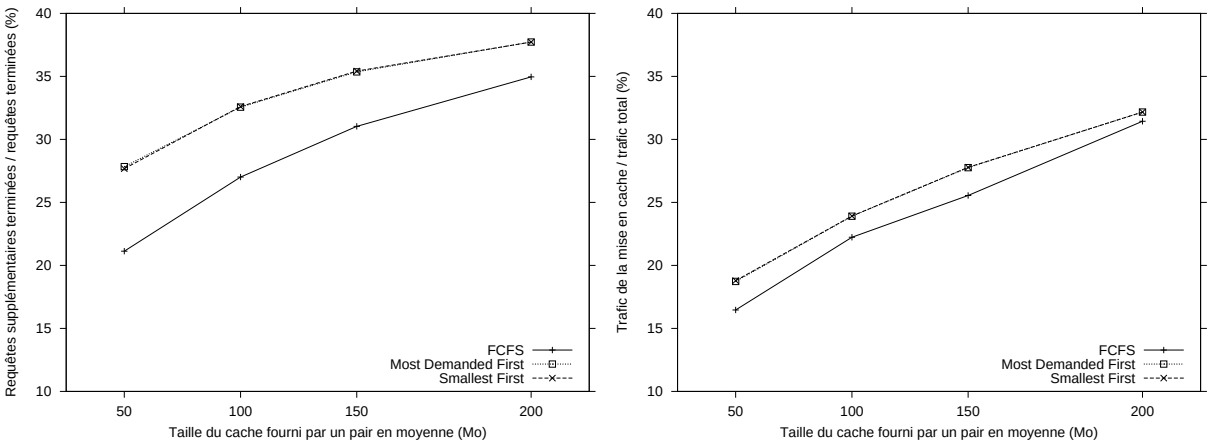
Sans surprise notable, les valeurs élevées de *Pcache* induisent un faible trafic alors que les petites valeurs génèrent un volume plus important de données à mettre en cache et induit donc un trafic supplémentaire assez conséquent (cf. Figure 3.18.b). Néanmoins, le trafic généré reste acceptable s'il est mis en comparaison avec le gain obtenu en terme de requêtes supplémentaires terminées. En effet, on remarque sur la Figure 3.18.a que la proportion de requêtes qui ont pu

se terminer s'accroît de manière remarquable avec la diminution de la valeur donnée à *Pcache*. À titre d'exemple, lorsque $Pcache = 0,01$, le nombre d'utilisateurs qui terminent leurs vidéos s'améliore de 31% tout en provoquant une augmentation acceptable du trafic dans le système, à savoir 24%. En effet, le trafic généré n'est pas très pénalisant pour les pairs, puisque la bande passante du réseau local sous-jacent est déjà disponible ; nous ne faisons que l'utiliser. Pour cette raison, nous pouvons affirmer que ces résultats sont très prometteurs et valident la pertinence de nos propositions.

Par ailleurs, la Figure 3.18 montre clairement que l'algorithme de tri employé pour choisir le suffixe à cacher en priorité n'a que peu d'impact sur les résultats lorsque $Pcache \in \{20\%, 40\%\}$. Ceci est dû au nombre restreint des threads de transmission autorisés par les pairs. En revanche, lorsque $Pcache \in \{1\%, 5\%, 10\%\}$, le fait de trier les requêtes de suffixe peut améliorer la disponibilité des données dans le système tout en générant le même trafic supplémentaire ou presque.

3.4.2.2 Impact de la taille du cache fourni par les pairs

Dans cette série d'expérimentations, nous avons mesuré l'impact de la taille de l'espace cache contribué par les pairs. Dans ce but, nous avons utilisé une petite valeur du paramètre $Pcache = 1\%$ pour avoir des suffixes plus volumineux et en plus grand nombre. Nous avons fait varier la taille du cache apporté par les pairs entre 50 et 200 Mo. Les résultats obtenus sont illustrés dans la Figure 3.19. On y voit le nombre supplémentaire de requêtes terminées grâce à la mise en cache des suffixes ainsi que le trafic supplémentaire qui en découle.



(a) Gain (b) Coût

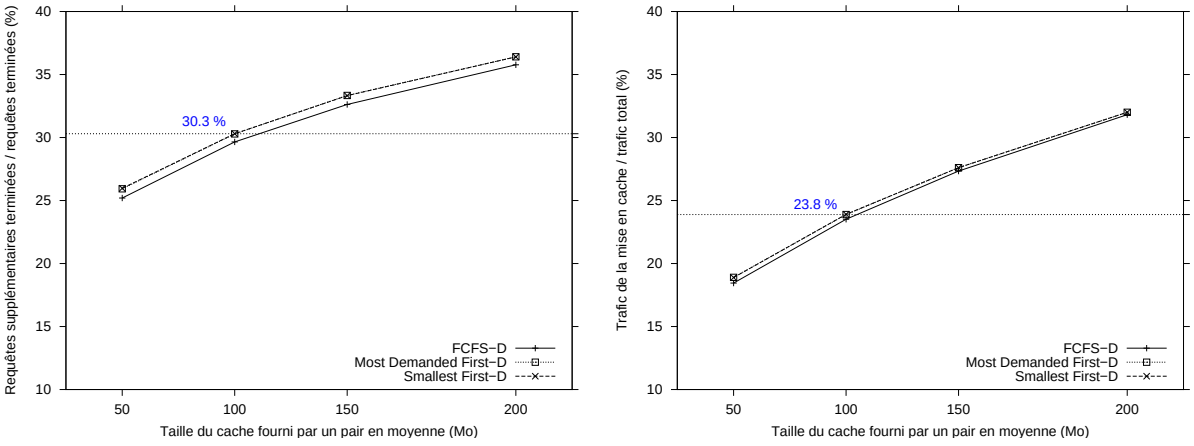
Figure 3.19 – Impact de la taille du cache fourni par les pairs sur la disponibilité des données.

Intuitivement, plus l'espace cache fourni par les pairs est grand, plus le nombre de requêtes de suffixe admises est élevé. Cette augmentation entraîne généralement la mise en cache de plus de données et accroît donc le nombre de requêtes terminées dans le système, améliorant par conséquent la disponibilité des données. Comme on peut le voir sur la Figure 3.19, si tous les pairs fournissent 200 Mo d'espace cache au système, le DVC formé par l'agrégation de ces espaces sera en mesure d'augmenter le nombre de requêtes terminées de 38%, le trafic supplémentaire généré sera toujours acceptable et de l'ordre de 32% du trafic total dans le système.

D'un autre côté, on remarque sur la même figure une nette amélioration des performances lorsque les requêtes de suffixe sont triées, par rapport à celles observées en employant le modèle FCFS pour traiter les requêtes de suffixe. Néanmoins, l'impact de la méthode de tri mise en place n'est guère visible. Ceci s'explique par le fait que nous ne mettons en cache que des portions de vidéos, de surcroît des suffixes, et qu'il ne s'agit pas de vidéos complètes.

3.4.2.3 Efficacité de l'approche adaptative face aux approches statiques

Nous nous sommes intéressés dans cette série d'expérimentations à étudier le comportement de notre politique de gestion de cache adaptative. Nous avons activé la fonctionnalité d'ajustement dynamique de $Pcache$ avec $Pcache_{step} = 0.002$ et en initialisant $Pcache$ à sa limite inférieure $Pcache_{inf} = 0.01$. Nous avons ensuite mesuré le rendement du DVC dans différentes configurations de la taille de cache fourni par les pairs ({50, 100, 150, 200} Mo). Les résultats obtenus sont représentés dans la Figure 3.20.



(a) Gain (b) Coût

Figure 3.20 – Performances de notre stratégie adaptative en fonction de la taille du cache fourni par les pairs.

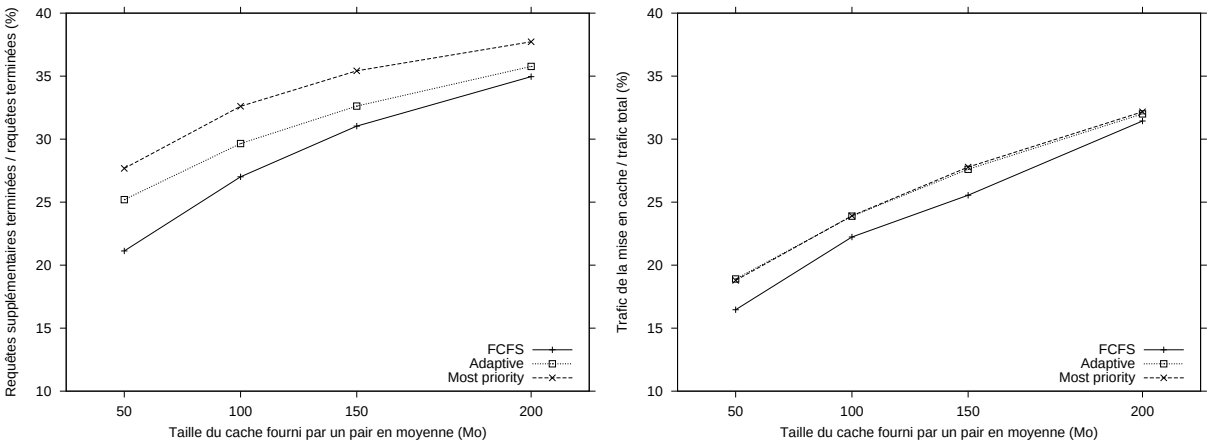
Le comportement affiché par le DVC lorsque notre stratégie adaptative est employée est comparable à celui observé lorsqu'une stratégie statique est utilisée avec une petite valeur de $Pcache$. En effet, sur la Figure 3.20, on constate trois similarités dans le comportement du DVC vis-à-vis des résultats présentés précédemment : (a) la méthode de tri employée a une influence très faible voir insignifiante ; (b) le nombre de requêtes terminées s'accroît avec l'augmentation de l'espace cache fourni par les pairs améliorant par conséquent la disponibilité des données dans le système ; (c) cette amélioration est accompagnée par une légère augmentation du trafic issu de la mise en cache des suffixes.

Sur la Figure 3.20, on remarque également que l'adaptation dynamique de $Pcache$ met sur un pied d'égalité, ou presque, les performances observées en employant le modèle FCFS avec celles mesurées lorsque les requêtes de suffixes sont triées. Nous en concluons qu'il n'est pas nécessaire de trier les requêtes de suffixe si notre stratégie adaptative est mise en place. Ce fait rend notre approche particulièrement intéressante à mettre en œuvre dans les systèmes de streaming P2P. En effet, notre politique a un temps d'exécution constant (cf. Algorithme 2),

alors que la complexité en temps des méthodes de tri SF et MDF est de l'ordre de $n \log n$ où n est le nombre de requêtes à trier.

D'ailleurs, la nature dynamique de notre stratégie lui apporte un autre avantage, non des moindres, à savoir la facilité de déploiement. Aucune configuration préalable de *Pcache* n'est nécessaire, puisqu'il s'adapte en temps réel, en fonction de la charge de mise en cache des suffixes et de la taille du DVC.

La Figure 3.21 compare les performances du DVC observées en employant notre stratégie, vis-à-vis d'autres politiques statiques configurées avec la limite inférieure de $Pcache_{inf} = 0.01$. Puisque les performances du DVC restent identiques, quelle que soit la méthode du tri employée, MDF ou SF, elles sont représentées sur la figure par une seule courbe intitulée *le plus prioritaire en premier* (MPF, pour Most Priority First). On y voit que les performances enregistrées par notre stratégie adaptative sont nettement supérieures à celles enregistrées par le modèle FCFS. Cependant, notre approche est légèrement en retrait par rapport à MPF. Néanmoins, le temps d'exécution constant de notre stratégie couplé avec son caractère dynamique nous permet de la recommander pour améliorer la disponibilité des données dans les systèmes de streaming P2P.



(a) Gain (b) Coût

Figure 3.21 – Comparaison entre FCFS, MDF et notre stratégie de gestion de cache adaptative.

3.4.3 Discussion

Les séries de tests présentées dans la section précédente, nous ont permis de quantifier le gain apporté par nos protocoles concernant la disponibilité des données dans un système de streaming P2P, mais aussi de mesurer leur coût d'utilisation en terme du trafic supplémentaire généré par ces protocoles. Comme l'attestent les résultats des simulations effectuées, nos protocoles améliorent considérablement la disponibilité des données sans pour autant induire d'énormes coûts.

Le fait que nos protocoles soient « pilotables » par un seul paramètre *Pcache* constitue un de leurs points forts. Nous avons alors fait varier ce paramètre de manière empirique, dans un premier temps. Nous avons noté une amélioration de la disponibilité des données d'environ 33% en moyenne pour une augmentation moyenne du trafic d'environ 23%.

Dans un deuxième temps, nous avons attribué au paramètre *Pcache* une valeur suffisamment petite (0,01) pour générer une charge conséquente de mise en cache des suffixes. Avec une telle configuration, nous avons pu mesurer l'impact de la taille de l'espace cache fourni par les pairs sur la disponibilité des données. Comme attendu, l'augmentation de cette taille entraîne naturellement une amélioration de la disponibilité des données.

Enfin, nous nous sommes plus particulièrement intéressé à l'ajustement automatique du paramètre *Pcache*. Notre stratégie adaptative a enregistré des performances comparables à celles obtenues en triant les requêtes de suffixe pour y répondre dans un ordre optimisé. L'avantage de notre approche réside dans le fait que son temps d'exécution est constant, alors que les méthodes de tri induisent une complexité en temps de l'ordre de $n \log n$, où n est le nombre de requêtes à trier. Notre méthode d'ajustement dynamique, par un système asservi, est donc tout à fait adaptée pour être employée dans les systèmes P2P, caractérisé par un dynamisme prononcé. Cependant, le pas de boucle de retour, qui permet de rendre le système « sensible » aux changements (plus tôt le système dispose de résultats de sortie, plus rapidement l'ajustement est effectif) a été fixé de manière empirique. Il serait alors intéressant, dans un future proche, de mener une étude théorique sur le système asservi utilisé (une analyse avancée impliquant une maîtrise de haut niveau en équations différentielles) pour déterminer la valeur optimale de ce pas.

3.5 Conclusion

Dans ce chapitre, nous avons mis l'accent sur le problème de la disponibilité des données dans les systèmes de streaming P2P, qui provient principalement du départ imprévisible des pairs. En effet, ce problème est tel qu'il est indispensable d'y remédier de manière aussi efficace et pertinente que possible. Nous avons donc modélisé cette problématique, et proposé une solution permettant de mitiger les effets de la déconnexion des pairs. Il s'agit de mettre en cache les suffixes des contenus en cours d'accès et qui seront vraisemblablement visionnés jusqu'à la fin. La mise en cache de ces suffixes limite le gaspillage des ressources du système P2P. Nous avons opté pour une architecture de cache distribué qui est alors formé par l'agrégation de petits espaces cache fournis par les pairs. Les suffixes y sont placés de manière à équilibrer la charge de streaming entre les pairs. D'ailleurs, nous avons élaboré une nouvelle stratégie adaptative pour gérer ce cache distribué. Cette stratégie s'appuie sur un système asservi pour ajuster la taille des suffixes à cacher, de sorte que le cache soit exploité au maximum. Comme l'attestent les résultats de nos expérimentations, notre approche est adéquate pour un environnement P2P.

Dans le prochain chapitre, nous portons un regard plus profond sur les systèmes de streaming P2P. Nous y décrivons l'architecture et les choix d'implémentation de notre système de streaming P2P : JStreaper.

Chapitre 4

JStreaper : un système de streaming Pair-à-Pair

Le succès des systèmes de streaming P2P n'est plus à démontrer et certains d'entre eux sont d'ores et déjà entrés dans la pratique du grand public. Comme nous l'avons vu dans le chapitre précédent, ces systèmes permettent aux usagers de partager des contenus multimédias, sans qu'un serveur dédié ne soit mis en place pour assurer le stockage et l'acheminement des contenus. La particularité de tels systèmes de partage P2P réside dans le fait que la transmission de contenus entre usagers est effectuée en mode streaming.

La grande majorité, si ce n'est la totalité, des systèmes de streaming P2P est conçue pour un déploiement à grande échelle impliquant, pour chaque contenu dans le système, la présence de plusieurs utilisateurs le partageant. Les usagers possédant un contenu donné coopèrent dans le streaming de celui-ci à un utilisateur désirant le visionner. Cette collaboration consiste pour chacun à prendre en charge le streaming d'une partie du contenu désiré, et cela de manière proportionnelle aux capacités de sa bande passante de sortie (cf. Section 3.2.2).

En revanche, de nos jours, la durée de connexion des utilisateurs à un système de streaming P2P est très courte, voire « instantanée ». Typiquement, un usager se connecte pour regarder un voire plusieurs contenus, puis une fois le visionnage terminé, il se déconnecte. Une telle utilisation peut être effectuée par une personne bloquée dans une file d'attente, par un voyageur à bord d'un avion, par un touriste à proximité de lieux des vacances, par un client d'une cafétéria ou par un étudiant au sein d'un campus pendant la pause. Ce type de comportement rend alors caduque l'hypothèse d'une connexion permanente et donc la disponibilité en de multiples exemplaires de tout contenu partagé. Ainsi, la transmission d'un contenu ne peut être assurée que par un seul usager le partageant.

Au travers de la conception et de la mise en œuvre de *JStreaper*, nous démontrons la faisabilité d'un système de streaming P2P où la transmission d'un contenu est assurée par un et un seul pair. À l'heure actuelle, les fournisseurs d'accès Internet n'offrent pas de connexion ADSL grand public permettant d'avoir la bande passante nécessaire pour assurer le streaming d'un contenu par un seul pair. Notre prototype trouve donc son domaine d'application dans les réseaux locaux à très haut débit. Cependant, cette limitation n'existera plus avec la démocratisation des connexions fibre optique, ce qui permettra l'utilisation de notre prototype à l'échelle d'Internet, sans aucune modification.

Par ailleurs, les utilisateurs de JStreaper peuvent être équipés de terminaux plus ou moins puissants, en termes de capacités de traitement et/ou d'affichage, car notre prototype permet de redimensionner les images des vidéos transmises en fonction de la configuration du terminal récepteur. Dans l'état actuel de nos connaissances, JStreaper est le premier prototype qui intègre le streaming P2P avec adaptation de contenus multimédias. En outre, le prototype JStreaper constituera une plate-forme de tests pour les protocoles développés au chapitre précédent. Une telle plate-forme permettrait, en effet, de mesurer les performances de ces protocoles avec des expérimentations en grandeur réelle.

La suite de ce chapitre est structurée de la façon suivante. Nous débutons par une présentation de l'architecture logicielle du prototype JStreaper et de ses différents modules. Les choix de mise en œuvre du prototype sont ensuite discutés, notamment en ce qui concerne le langage et les bibliothèques de programmation utilisés. Puis, nous décrivons les détails du fonctionnement interne de chaque module de JStreaper. Enfin, nous présentons les résultats des tests préliminaires de notre prototype avant de conclure ce chapitre.

4.1 Architecture logicielle

Le composant principal de notre système est appelé *Streaper*[1] : un pair qui est capable de partager des contenus multimédias en mode streaming. Notre prototype nommé *JStreaper*, comme tout système de streaming P2P, ne met pas en place de serveur dédié au stockage et au streaming des contenus audiovisuels ; ces services sont fournis par les Streapers eux-mêmes. Toutefois, un annuaire centralise le service de recherche de ressources dans le système. Cet annuaire, appelé *Dispatcher*, contient des informations relatives aux Streapers connectés et aux contenus mis à disposition par ces derniers. Notre prototype peut donc être qualifié de système pseudo pair-à-pair, dans le sens des définitions présentées dans la Section 3.1.1.

L'architecture globale de JStreaper est illustrée par la Figure 4.1. On y voit deux types d'entités : un Dispatcher et plusieurs Streapers. Un Streaper est un logiciel permettant aux utilisateurs de se connecter au système afin de partager des contenus multimédias et/ou de regarder les contenus partagés par d'autres usagers. Un Streaper ne présente pas un comportement de serveur de streaming, c'est-à-dire la prise en charge d'un nombre conséquent de streams simultanément. Néanmoins, il est en mesure de redimensionner les images des vidéos transmises en fonction des préférences du Streaper récepteur, celles-ci étant dictées par les capacités de traitement et d'affichage de ce dernier. Dans l'état actuel de nos connaissances, JStreaper est le premier prototype qui intègre une architecture P2P avec adaptation de la résolution des vidéos en fonction de la configuration du terminal récepteur.

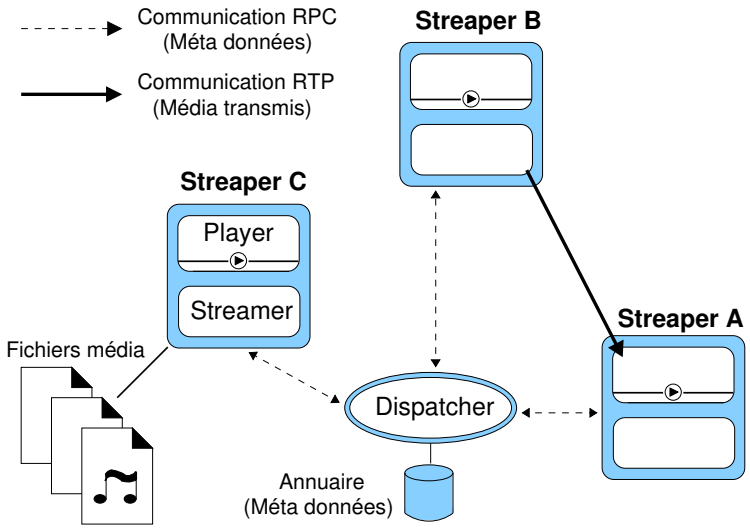


Figure 4.1 – Vue générale de l'architecture de JStreaper.

Notre souci premier, lors de la conception de JStreaper, fut sa modularité. Pour cette raison, nous avons adopté un découpage fonctionnel permettant d'étendre, d'optimiser et/ou de tester un module sans remettre les autres en cause. Par la suite, nous décrivons un scénario illustrant le fonctionnement du système JStreaper. Puis, nous mettons en relief les différentes techniques de gestion intrinsèques à chaque module ainsi que les tâches qui leur sont déléguées.

1. Streaper est l'acronyme de stream et peer.

4.1.1 Scénario d'utilisation

Un cas typique d'usage de JStreaper, avec trois usagers A, B et C (voir Figure 4.1) se résume en six étapes comme suit :

1. Le Dispatcher démarre et se met en attente d'utilisateurs souhaitant rejoindre le système via des Streapers.
2. Les Streapers (A, B et C) se connectent au Dispatcher et y enregistrent leur adresse IP, leurs fichiers audiovisuels partagés et le nombre maximum de flux simultanés autorisés.
3. Le Streaper A interroge le Dispatcher afin de récupérer la liste de tous les contenus partagés par les autres Streapers connectés au système, c'est-à-dire ceux partagés par les Streapers B et C. Il est à noter que cette liste ne contient pas d'informations relatives aux usagers, leur anonymat reste par conséquent préservé.
4. Une fois que l'utilisateur du Streaper A a choisi un contenu, ce Streaper envoie une requête pour visionner le contenu sélectionné au Dispatcher. Ce dernier désigne alors un Streaper, partageant le contenu en question, pour prendre en charge le streaming de ce contenu vers le Streaper A. Supposons que le Dispatcher a nommé le Streaper B pour assurer cette tâche. Le Dispatcher envoie au Streaper A des informations lui permettant d'établir une communication directe avec le Streaper B.
5. Le Streaper B demande au Streaper A ses préférences en matière de qualité audiovisuelle, celle-ci est dictée par les capacités de traitement et d'affichage de la machine sur laquelle est installé le Streaper A. En fonction de ces préférences, le Streaper B configure un *filtre* dont le rôle est d'adapter en conséquence le stream souhaité . Notons que la configuration du filtre d'adaptation est antérieure au démarrage du processus de streaming, et elle ne change pas durant ce processus.
6. Le Streaper B commence la transmission du contenu adapté au Streaper A, c'est-à-dire de Streaper à Streaper.

4.1.2 Modèle de communication

Dans JStreaper, nous avons adopté deux modes de communications : le streaming et l'appel de procédures distantes [BN84] (RPC). Un protocole de transmission temps réel, comme RTP (cf. Section 1.3.2.2), est utilisé pour l'acheminement des données audiovisuelles entre Streapers. Toutes les autres communications entre Streapers sont effectuées en invoquant des procédures à distance (RPC). Aussi, les communications entre les Streapers et le Dispatcher obéissent au même modèle. Nous avons retenu ce modèle de communications puisqu'il facilite la programmation de notre système.

4.1.3 Dispatcher

Dans cette première version de JStreaper, et dans l'objectif de simplifier sa conception, le Dispatcher est une instance centralisée, installée sur une machine dédiée que nous considérons fiable. Toutes les informations relatives aux usagers connectés au système et leurs contenus partagés y sont inscrites. Chaque usager est reconnu par un identifiant unique tout en préservant son anonymat. Le Dispatcher conserve l'adresse IP de la machine utilisée par l'utilisateur. Quant

aux contenus partagés par un usager, le Dispatcher maintient principalement les noms identifiant ces contenus, leur taille, leur format, leur débit ainsi que l'identifiant des utilisateurs qui les partagent. Le Dispatcher possède donc une vue globale de l'état du système.

Lorsqu'un utilisateur souhaite connaître la liste des contenus disponibles dans le système à un moment donné, il interroge le Dispatcher par le biais de son Streaper. Après avoir récupéré la liste des contenus, l'utilisateur peut sélectionner un contenu l'intéressant. Suite à cette sélection, le Streaper envoie une requête au Dispatcher afin que ce dernier lui alloue un Streaper chargé d'assurer le streaming du contenu choisi. Le Dispatcher recherche alors les informations relatives au Streaper partageant le contenu demandé [2]. Il vérifie notamment que le nombre de flux de transmission simultanés sur ce Streaper n'a pas atteint le seuil autorisé. Un port de communication ainsi que l'adresse IP du Streaper partageant la vidéo sont ensuite envoyés au Streaper qui est à l'origine de la requête. Ce dernier peut désormais, en se servant de ces deux éléments, établir une communication directe avec le Streaper chargé de lui transmettre le contenu désiré.

4.1.4 Streaper

Un Streaper est un logiciel destiné à être installé sur la machine de chaque utilisateur souhaitant se connecter au système. Il permet aux usagers dans le système de partager des contenus multimédias et de regarder les contenus partagés par d'autres, le tout en mode streaming. Il est principalement composé de deux modules : un lecteur audiovisuel et un moteur de streaming, nommés respectivement *Player* et *Streamer*. Ces deux modules, que nous décrivons dans les sous-sections suivantes, sont capables de fonctionner de manière simultanée.

4.1.4.1 Lecteur multimédia

Comme son nom l'indique, ce module permet de décoder et de restituer des données audiovisuelles. Celles-ci peuvent être issues d'un fichier local ou alors d'un stream dont l'origine est l'un des Streapers présents dans le système. L'architecture interne du Player est conçue à l'image de celle détaillée dans la Section 1.3.1. Elle comprend une interface graphique (GUI) permettant aux usagers d'interagir sur les streams restitués, d'augmenter ou de diminuer le volume, de mettre en pause ou de reprendre la restitution, etc. Via cette interface, les utilisateurs peuvent également envoyer des requêtes au Dispatcher, pour récupérer la liste des contenus disponibles ou pour demander l'accès à un contenu donné. Le Player inclut aussi plusieurs codecs (cf. Section 1.1.2) afin qu'il puissent décoder des streams de différents formats.

4.1.4.2 Moteur de streaming

Ce module est le composant qui permet aux utilisateurs du système de partager des contenus multimédias en mode streaming. Il est responsable de la transmission d'un contenu audiovisuel partagé, vers le Player d'un autre usager souhaitant le regarder.

2. Il se peut que plusieurs Streaper partagent un même contenu, sous différents noms par exemple. Dans ce cas, il incombe au Dispatcher d'appeler une *procédure de sélection* afin d'en choisir « le plus convenable » pour assurer la charge de streaming. Cependant, de telles considérations n'ont pas été prises en compte dans cette première version de JStreaper, puisqu'elles sortent du cadre de cette thèse.

Les usagers peuvent être munis d'équipements très hétérogènes en termes de capacité de calcul et/ou d'affichage. En outre, le décodage d'une vidéo basse qualité demande moins de capacité de calcul qu'une vidéo haute qualité. Il est donc judicieux que les machines plutôt puissantes adaptent les streams en fonction des capacités du terminal chargé de les restituer. En effet, présenter sur un assistant personnel (PDA) une vidéo encodée pour être visionnée sur un écran de station de travail est souvent inefficace parce qu'un PDA dispose d'un écran plus petit. Le PDA doit redimensionner la vidéo alors qu'il ne possède pas la capacité de calcul permettant d'effectuer cette opération à la volée.

Un moteur de streaming installé sur une machine puissante peut effectuer une adaptation à la volée durant le streaming d'un contenu à un Streaper chargé de le restituer. Il s'agit d'une adaptation spatiale en fonction de préférences définies à priori par l'utilisateur de ce Streaper. Ces préférences définissent généralement la qualité audiovisuelle souhaitée en terme de résolution maximale qu'une machine est capable de restituer aisément. Avant le démarrage d'un processus de streaming, le Streamer qui en est chargé consulte les préférences inscrites chez le Player destinataire. En fonction des préférences reçues, le moteur de streaming configure alors un filtre adaptatif à appliquer sur les données audiovisuelles lors de la transmission.

À l'opposé d'un serveur de streaming, un streamer ne prend en charge qu'un nombre restreint de streams simultanés. Ceci est dû d'une part aux capacités limitées des ordinateurs personnels face à celles des serveurs dédiés au streaming. D'autre part, chaque utilisateur autorise un nombre restreint de flux simultanés sur sa machine, afin qu'elle ne soit pas submergée par la charge du streaming. Nous adoptons donc une architecture multithreadée dont la taille de l'ensemble des processus légers (*thread pool*) réservés pour le streaming de contenus audiovisuels est fixée par l'usager. Cette architecture est illustrée par la Figure 4.2 où le moteur de streaming du Streaper Y possède au maximum quatre threads de transmission, dont deux sont inactifs. Le Streaper Y peut donc accepter de prendre en charge deux streams supplémentaires, ceux-ci provenant de ses contenus partagés.

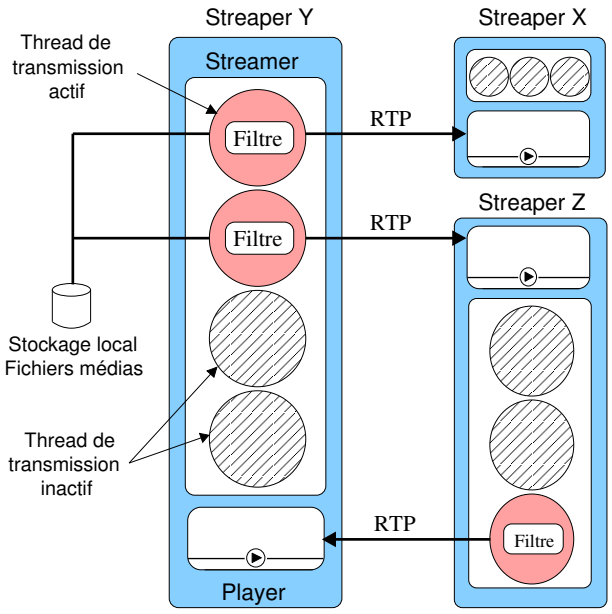


Figure 4.2 – Architecture multithreadée dans un Streamer.

4.2 Choix d'implémentation

JStreaper est destiné à être déployé dans un environnement P2P. L'éventail des services offerts aux Streapers connectés est caractérisé par un aspect fortement distribué, ces services étant fournis par les Streapers eux mêmes. D'ailleurs, les Streapers sont appelés à communiquer dans un environnement très hétérogène. En effet, les utilisateurs de JStreaper peuvent posséder divers types d'équipements (ordinateurs portables, assistants personnels, etc.) fonctionnant sous différents systèmes d'exploitation. Nous nous sommes donc attachés à ce que le code de JStreaper soit portable, en vue de faciliter son déploiement [3].

Aussi, comme cela a été mentionné dans la section précédente, JStreaper emploie de manière interne plusieurs modes de communication, à savoir l'appel de procédure à distance (RPC) ainsi que la transmission de données audiovisuelles en temps réel (RTP). Afin de faciliter la mise en œuvre de la communication au sein de notre prototype, l'utilisation de bibliothèques appropriées nous a semblé nécessaire. En outre, des bibliothèques permettant la manipulation des données multimédias sont également indispensables pour l'adaptation ce celles-ci.

Par ailleurs, nous avons décidé de bâtir le prototype JStreaper sur l'utilisation de processus légers (*threads*). Ceci est motivé d'une part par le fait que d'un point de vue système, un thread consomme moins de ressources à sa création qu'un processus ordinaire. D'autre part, les communications basées sur une architecture émetteur/récepteur sont bien adaptées à l'utilisation de threads.

Au vu de cette analyse des besoins de notre système de streaming P2P, notre choix s'est porté sur *la technologie Java*™. En favorisant l'indépendance du logiciel vis-à-vis du matériel, cette technologie permet, en effet, de surmonter les écueils inhérents à la création des applications distribuées. Dans la suite de cette section, nous décrivons cette technologie en mettant en lumière les raisons qui nous ont conduit à l'adopter.

4.2.1 Technologie Java

La technologie Java est composée du langage de programmation Java et de son environnement d'exécution, la JVM. La large diffusion de ce couple a permis à Java de s'imposer comme une solution prédominante de nos jours. Les caractéristiques les plus importantes de la JVM et du langage Java sont brièvement décrites dans les paragraphes suivants.

4.2.1.1 Machine Virtuelle Java

Une machine virtuelle est, en quelque sorte, une machine abstraite qui permet de masquer au niveau applicatif les spécificités de la machine, et plus précisément son architecture sous-jacente et son système d'exploitation. Elle exécute, comme une vraie machine, un langage assembleur qui lui est propre, on parle ici de *byte code*. L'indépendance du byte code vis-à-vis de la plate-forme matérielle l'exécutant, le dote d'une portabilité absolue sur toutes les plates-formes supportant son environnement d'exécution.

La machine virtuelle Java (JVM) est sans aucun doute la plus connue et la plus utilisée actuellement [LY99]. Elle a été créée en marque déposée par Sun Microsystems en 1995. Bien

3. Le déploiement d'une application distribuée consiste en la diffusion de programmes exécutables, leur installation puis leur configuration sur les sites de leur future exploitation.

que la JVM garantisse la portabilité et la sécurité des applications, elle ralentit légèrement l'exécution ce celles-ci, comme toute machine virtuelle. La réputation de lenteur conférée à Java remonte à ses origines bien que l'amélioration constante des JVM permette à l'heure actuelle d'obtenir des performances très proches de celles d'une application native. D'ailleurs, la gestion automatique de la mémoire facilite davantage la tache des programmeurs en leur permettant de se concentrer sur la logique de traitement de l'application. Les caractéristiques de la JVM qui nous semblent les plus intéressantes sont :

La sécurité : La JVM rend impossible certains types d'attaque, comme la surcharge de la pile d'exécution, l'endommagement de la mémoire extérieure à son propre espace de traitement ainsi que la lecture ou l'écriture de fichiers sans permission. D'ailleurs, à l'aide de la notion de classe signée numériquement, il est possible d'identifier l'auteur d'une classe donnée et de déterminer ainsi l'ampleur des privilèges accordés à cette classe en fonction de la confiance envers son auteur. Dans un cadre P2P, garantir aux utilisateurs que le logiciel mis à leur disposition n'endommage en aucun cas leur machine, est un des avantages majeurs de la technologie Java.

Les interpréteurs JIT : Les JVM, à l'exception de celles développées pour les téléphones portables, fournissent des interpréteurs JIT (« Just-In-Time »), permettant d'obtenir de bonnes performances. Celles-ci sont dues à la mémorisation de l'interprétation du byte code en code natif, à l'issue du premier appel. En outre, Sun Microsystems a introduit, depuis la version 1.5, des machine virtuelles optimisées nommées *HotSpot*. Ces dernières améliorent davantage les performances des interpréteurs JIT en effectuant plusieurs tâches supplémentaires, comme la découverte des goulots d'étranglement ou la recompilation en code natif des parties de code fréquemment utilisées. L'utilisation d'une JVM HotSpot est particulièrement intéressante dans le cadre du développement du JStreaper puisqu'elle permet d'accroître les performances lorsque la durée de connexion des Streapers augmente.

Le ramasse-miettes automatique : Ce service facilite le travail du programmeur en le déchargeant de la gestion de la mémoire. Toutefois, une telle automatisation est moins rapide qu'une gestion manuelle et est plus consommatrice en ressources système, en particulier en espace mémoire. Cependant, le surcoût en consommation de mémoire est devenu négligeable de nos jours, puisque même les terminaux légers possèdent un espace mémoire relativement grand. On peut conduire le même raisonnement pour des processeurs performants.

4.2.1.2 Java, le langage

Java est un langage de programmation strictement [4] orienté objet mis au point par Sun Microsystems. La maturité du langage Java, ainsi que la richesse des API l'accompagnant ont beaucoup contribué à lui accorder sa place parmi les langages de programmation les plus populaires [GJSB05]. Par ailleurs, l'une des raisons principales du succès de Java est la facilité qu'il procure lors de la manipulation de threads, ce qui est particulièrement utile dans une architecture fortement multithreadée, telle que JStreaper.

De nombreuses bibliothèques sont disponibles pour Java. En standard, il est livré avec une API évoluée comportant plus de 3000 classes minutieusement étendues, testées et éprouvées. Ces classes couvrent un spectre très étendu allant de la construction des interfaces utilisateur (AWT, Swing) à la gestion de bases de données (JDBC) en passant par une couche réseau

4. Contrairement à C++ par exemple, on ne peut que faire de la programmation orientée objet avec Java.

complète (Socket, RMI). Ci-après, nous présentons en particulier les deux bibliothèques de programmations qui nous intéressent le plus dans le cadre du développement de JStreaper, à savoir RMI qui permet d'appeler des procedures à distance et JMF qui permet de manipuler et de transférer des données audiovisuelles en temps réel.

4.2.2 Java RMI

Afin de faire communiquer les Streapers entre eux, ou avec le dispatcher, nous utilisons RMI qui est une API livrée avec Java en standard. Cette API permet d'appeler des méthodes distantes (RPC), et plus précisément d'invoquer les méthodes des objets Java distants, c'est-à-dire des objets instanciés sur une autre JVM que celle de la machine locale. Les appels distants se font de manière transparente, comme si l'objet était sur la JVM de l'ordinateur local. La couche de communication ainsi que la localisation de l'objet distant sont masquées.

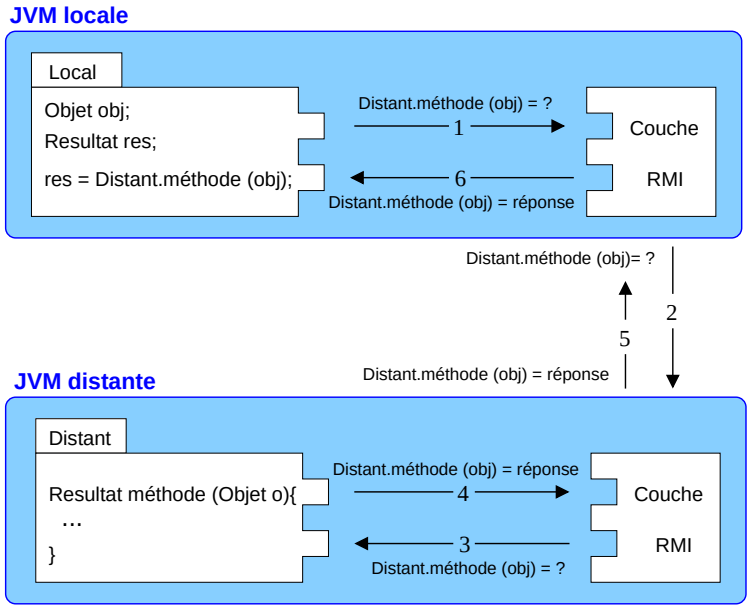


Figure 4.3 – Fonctionnement de RMI.

4.2.3 Java Media Framework (JMF)

Pour manipuler les données multimédias, JStreaper repose sur l'environnement JMF, dans sa version 2.1.1e. Il s'agit d'une bibliothèque développée par Sun Microsystems, et dont le but est de faciliter l'implémentation des applications multimédias en Java. Nous avons utilisé JMF pour de multiples raisons. Elle est modulaire et simplifie donc la combinaison de composants logiciels. Des versions optimisées pour différentes plates-formes sont également disponibles. Elle est assez complète et fournit la plupart des composants dont nous avons besoin. En effet, cette bibliothèque comporte les fonctionnalités nécessaires à la restitution, à la transmission et au traitement de données audiovisuelles. Notons que le JMF fournit également des fonctionnalités relatives à la capture et au stockage des données multimédias.

4.2.3.1 Restitution des données audiovisuelles

La restitution des données avec JMF est faite de manière similaire au fonctionnement d'un système audiovisuel utilisant des dispositifs de la vie courante : un DVD, un lecteur DVD, des haut-parleurs et un téléviseur. La classe `DataSource` permet, tout comme un DVD, d'encapsuler des données audiovisuelles. La classe `Player`, à l'image d'un lecteur DVD, fournit les mécanismes de contrôle et de restitution des données multimédias en temps réel. La Figure 4.4 illustre parfaitement cette ressemblance. Il est à noter que la source des données peut être distante, à condition que ces données soient transmises de manière conforme au protocole RTP. Notons également que JMF fournit de nombreux codecs permettant de restituer des données audiovisuelles de différents formats. Le lecteur peut se référer à [jmf] pour une liste complète des formats supportés.

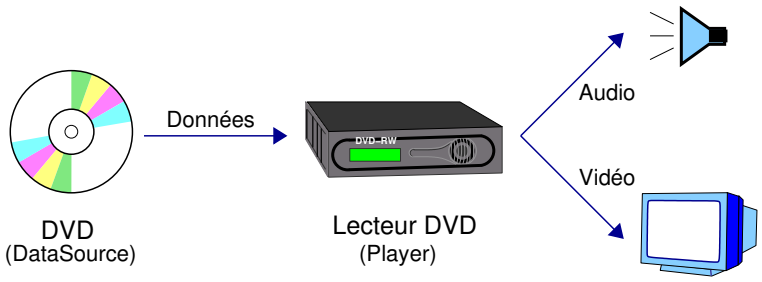


Figure 4.4 – Restitution de données audiovisuelles dans JMF.

4.2.3.2 Traitement des données audiovisuelles

Un flux multimédia est composé de l'assemblage d'un ensemble de flux médias, pouvant contenir de l'audio ou de la vidéo. L'unité de traitement dans JMF est le *processeur* dont l'architecture est représentée par la Figure 4.5. Un processeur est un canal de traitement de flux multimédias à trois étapes. Une qui assemble les données (paquets) reçues en entrée et démultiplexe les flux médias, audio et vidéo. Une autre pour le traitement de chaque stream en utilisant soit des codecs spécifiés à l'initialisation du processeur, soit des effets qui sont des spécialisations de codecs pouvant être introduites pour adapter le flux à un usage particulier. Une dernière étape, facultative, pour le multiplexage et l'encodage des flux médias résultants en paquets destinés à l'émission sur le réseau en utilisant le protocole RTP (voir le paragraphe suivant).

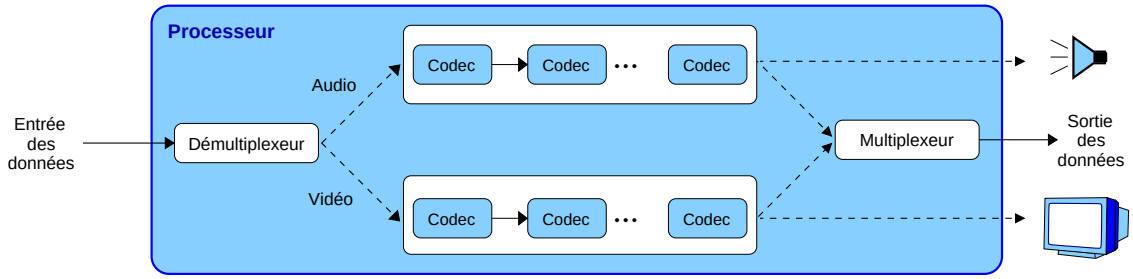


Figure 4.5 – Architecture d'un processeur JMF.

4.2.3.3 Transmission des données audiovisuelles

L'environnement JMF est doté d'une bibliothèque permettant la diffusion en continue (streaming) de données audiovisuelles via le protocole RTP (voir Figure 4.6). Les données transmises peuvent être ensuite restituées, chez le récepteur, à l'aide de la classe `Player` présentée précédemment, ou alors par le biais d'un logiciel quelconque capable de restituer des flux RTP standards. D'ailleurs, JMF permet la gestion de sessions multimédias, via la classe `RTPManager`. Cette dernière maintient des statistiques issues d'une session et permet de la contrôler conformément au protocole RTCP. Notons enfin que JMF est extensible. On peut en effet implémenter le découpage des données audiovisuelles en paquets (*empaquetage*), ainsi que l'extraction de celles-ci des paquets (*dépaquetage*) de manière spécifique à un codage particulier.

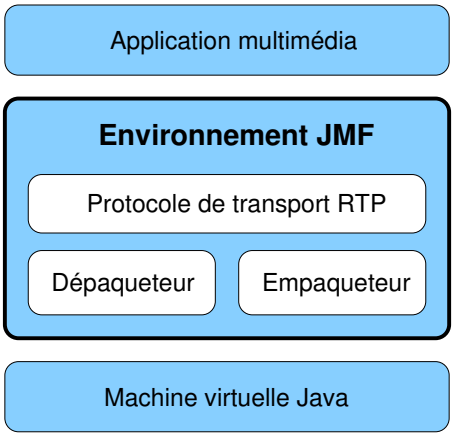


Figure 4.6 – Application utilisant l'environnement JMF.

4.3 Réalisation

JStreaper est implémenté en Java, ce qui le dote d'une portabilité sur toute plate-forme compatible Java et ayant la bibliothèque JMF d'activée. Accessoirement, le principe de « écrit une fois et exécuté partout », qui caractérise tout code Java, rend relativement aisée la distribution de JStreaper. D'ailleurs, l'aspect « pur-Java » de notre prototype le dote d'une énorme commodité de diffusion, puisque le processus d'installation d'un Streaper se réduit à la simple copie d'un fichier JAR.

Notre principal objectif, lors de la mise en œuvre de JStreaper, était de fournir une implémentation efficace de ces différents modules, tout en garantissant une bonne ergonomie à ces interfaces graphiques. Dans ce qui suit, nous détaillons l'implémentation des différents composants de notre prototype, à savoir le Dispatcher, le Player et le Streamer.

4.3.1 Dispatcher

Le Dispatcher est un module qui centralise des fonctionnalités minimales. Il gère un fichier XML qui concrétise l'annuaire contenant les informations relatives aux usagers et à leurs contenus partagés (voir un exemple de ce fichier ci-après). Lorsqu'un utilisateur désire se connecter

au système, son Streaper invoque à distance la méthode `log-in` sur le Dispatcher. Cette méthode attribue au Streaper un identifiant unique et enregistre dans le fichier XML les informations concernant cet usager, en l'occurrence son adresse IP, ses contenus partagés, et le nombre de threads de transmission disponibles à un moment donnée. Ce dernier paramètre permet au Dispatcher de savoir si un Streaper est en mesure de prendre en charge le streaming d'un de ces contenus partagés, autrement dit, si le nombre de threads de transmission disponibles (en veille) sur ce Streaper n'est pas nul.

Le contenu du fichier XML est actualisé à chaque fois que l'état d'un Streaper est modifié : (dé)connexion du Streaper ; changement dans les fichiers partagés et/ou dans le nombre de threads de transmission disponibles. De cette manière, l'annuaire du Dispatcher, qui est incarné par le fichier XML, est maintenu à jour pour toute consultation.

Exemple d'une liste des Streapers : fichier XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <streaper-list>
    <streaper>
      <id>11</id>
      <ip>10.11.11.25</ip>
      <free-threads>3</free-threads>
      <sharing-list>
        <shared-file>
          <name>Knick-Knack.mpg</name>
          <format>mpeg1</format>
           ...
        </shared-file>
        <shared-file> ... </shared-file>
        ...
      </sharing-list>
    </streaper>
    ...
    <streaper>
      <id>12</id>
      <IP>10.11.11.40</IP>
      <free-threads>5</free-threads>
      <sharing-list>
        <shared-file>
          <name>For_The_Birds.mov</name>
          <format>quick-time</format>
           ...
        </shared-file>
        ...
      </sharing-list>
    </streaper>
  </streaper-list>
```

Comme mentionné dans la Section 4.1, nous avons identifié deux types de requêtes dont les objectifs sont différents. Il s'agit pour le premier type de consulter la liste des contenus disponibles dans le système à un moment donné. En réponse à une telle requête, une liste des contenus disponibles au moment de la demande est générée. Quant aux requêtes du deuxième type, elles demandent toujours des informations permettant d'établir une communication directe avec le Streaper partageant le contenu désiré. En vue de répondre à une telle requête, le Dispatcher identifie tout d'abord le Streaper qui partage le contenu demandé. La réponse à la requête est alors constituée de l'adresse IP du Streaper partageant le contenu ainsi que d'un port de communication choisi à partir d'une plage définie par l'utilisateur de ce Streaper.

4.3.2 Player

Le Player est le composant qui permet la restitution de données audiovisuelles, que la provenance de celles-ci soit un fichier local ou un flux audiovisuel d'un Streaper dans le système. Or, comme cela a été mentionné dans la Section 2.1.2.2, lors d'une diffusion en continu d'un flux multimédia, une désynchronisation peut se produire entre le flux audio et les images qui lui sont associées. Dans de tels cas, le Player est capable de resynchroniser les deux flux et par conséquent peut garantir une qualité de service acceptable dans notre système (voir Figure 4.7).

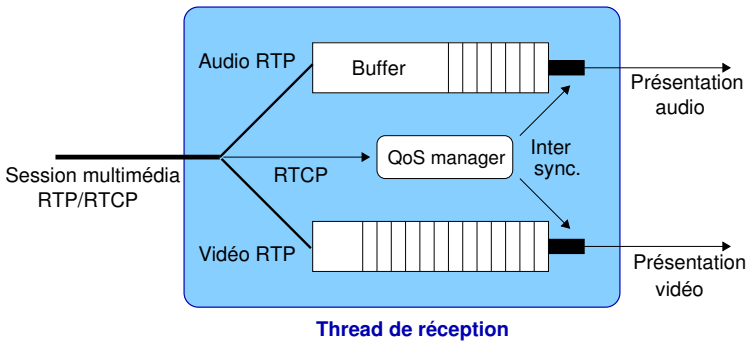


Figure 4.7 – Synchronisation entre une vidéo et son discours audio dans un thread de réception.

L'accès aux contenus partagés dans le système est rendu possible via une boîte de dialogue agréable à utiliser aussi bien par le grand public que par des utilisateurs « experts » (voir Figure 4.8). Au travers cette fenêtre, un utilisateur peut consulter la liste des contenus disponibles

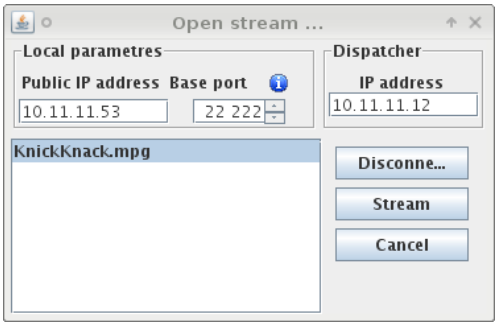


Figure 4.8 – Fenêtre de dialogue pour l'ouverture d'un stream.

dans le système à un moment donné. Aussi, il peut demander l'accès à un contenu particulier, en spécifiant le port de communication à utiliser pour le lui transmettre. Rappelons que ces requêtes, dont le destinataire est le Dispatcher, sont mises en œuvre par des invocations de méthodes à distance sur ce dernier. Une fois qu'une connexion est établie entre le Player et le moteur de streaming élu pour assurer la transmission du contenu désiré, un gestionnaire de synchronisation est créé, par le Player, pour gérer les flux audiovisuels reçus.

Notons enfin que l'interface graphique attachée à ce module est issue de la bibliothèque JMF, ce que lui confère une bonne ergonomie permettant d'effectuer toutes les interactions typiques sur les flux audiovisuels. Celles-ci comprennent la lecture, la mise en pause, l'arrêt, la reprise ainsi que l'augmentation ou la diminution du volume. D'ailleurs, le passage en mode plein écran et les raccourcis clavier dans notre Player ont été implémentés à l'images de ceux adoptés par le célèbre lecteur multimédia VLC (cf. Section 1.3.1.4). Un aperçu de la fenêtre principale du Player est donné dans la Figure 4.9.

Figure 4.9 – Fenêtre principale d'un Player.

4.3.3 Streamer

Comme nous l'avons vu dans la Section 4.1, le Streamer est le module responsable de la diffusion en continu des données audiovisuelles entre Streapers. Ce moteur de streaming possède une architecture multithreadée illustrée par la Figure 4.2. Un des objectifs que nous nous sommes imposés lors de l'implantation du Streamer est de le gratifier d'une interface graphique (GUI) permettant aux usagers de le configurer facilement selon la capacité de leur matériel. Cette interface s'affiche au lancement du Streamer, par le chargeur des classes de la machine virtuelle locale (`ClassLoader`). L'usager est ainsi appelé à paramétrer son moteur de streaming en fonction de ses besoins. En fixant le nombre maximal de flux simultanés, un usager peut éviter que son matériel ne soit submergé par la charge de streaming. Le Streamer est ensuite démarré et attend que des tâches de streaming lui soient attribuées par le Dispatcher. Celles-ci se rapportent à la transmission de l'un de ces contenus partagés vers un autre Streaper dans le système.

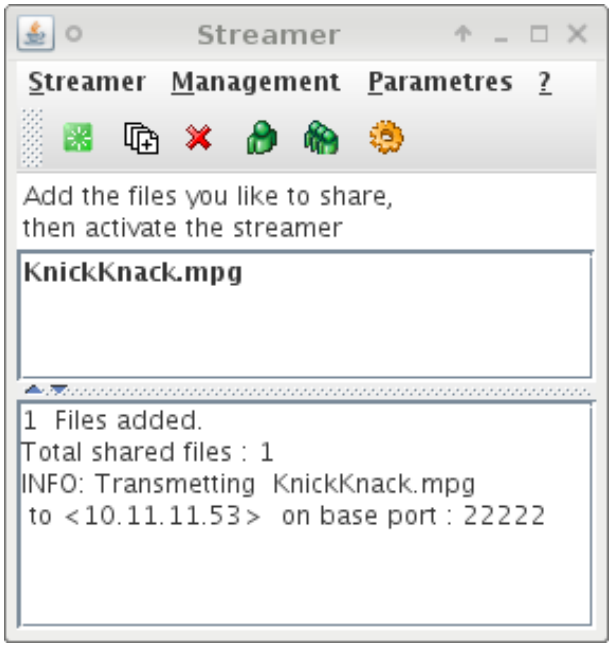


Figure 4.10 – Fenêtre principale d'un Streamer.

L'architecture multithreadée du Streamer a été mise en œuvre à l'aide d'une classe introduite dans Java à partir de sa version 1.5, à savoir la classe `ThreadPoolExecutor`. Via une interface graphique, un utilisateur peut définir la taille maximale autorisée du thread pool et limiter par conséquent le nombre de transmissions accueillies simultanément par sa machine. En outre, l'usager peut spécifier les ports à utiliser pour établir les communications avec d'autres Streapers.

L'interface de configuration permet également aux usagers de retirer et/ou ajouter des fichiers multimédias dans leurs contenus partagés à tout moment. Aussi, elle leur permet de superviser l'accès à leurs contenus en temps réel. La Figure 4.10 montre la fenêtre principale du Streamer. On y voit que le fichier partagé « `KnickKnack.mpg` » est en cours d'accès par un autre Streaper dont l'adresse IP est `10.11.11.53`.

Comme cela a été mentionné précédemment, après avoir accepté une tâche de streaming, le Streamer prépare un filtre adaptatif à appliquer sur les données audiovisuelles lors de la transmission au Streaper destinataire. Nous nous sommes limités à l'adaptation des données visuelles. Par le biais des classes `Processor`, `QualityControl` et `VideoFormat` de la bibliothèque JMF (cf. Section 4.2.3), une adaptation de la taille de chaque image d'une vidéo est effectuée en fonction des préférences du Streaper destinataire. Ces préférences sont envoyées par ce dernier lors de sa demande et elles se rapportent à la résolution maximale souhaitée. Il est à noter que les diverses configurations du filtre produisent des flux de différentes résolutions visuelles et entraînent donc des trafics à des débits différents. Rappelons également que la configuration du filtre d'adaptation est effectuée préalablement à l'ouverture d'une session multimédia et ne change pas tout au long de la session.

Après la configuration d'un filtre adapté aux préférences du Streaper demandeur, un thread est alloué pour assurer le streaming du contenu désiré. Ce thread prend en charge la mise en paquets des données à transmettre à l'aide de la classe `RTPManager` qui fait partie de la bibliothèque JMF. Le flux RTP résultant est envoyé au Streaper destinataire en utilisant son

adresse IP et un port de communication. Ces coordonnées sont communiquées au moteur de streaming au moment de la demande. Le thread de transmission n'est libéré pour rejoindre le pool qu'à la fin de la tâche de streaming, que cette fin soit due à la rétractation de la part de l'utilisateur demandeur ou à l'arrivée au terme du contenu demandé.

4.4 Tests préliminaires

Nous avons mesuré l'utilisation du processeur (CPU) d'un ordinateur portable en fonction du nombre de threads de streaming exécutés simultanément par un Streamer. L'ordinateur est équipé d'un processeur Intel Core 2 Duo à 1,5 GHz avec 2 Go de mémoire, son système d'exploitation est un GNU/Linux Debian (Lenny), la machine virtuelle Java utilisée est une HotSpot Server (version 1.6.0_10) avec la version 2.1.1e de la bibliothèque JMF. Toutes les mesures ont été effectuées avec le même fichier vidéo dont le format est MPEG-1 avec une taille de 352x240 points et une cadence de 30 images/seconde. Chaque thread de transmission adapte ce fichier à la moitié de sa résolution. Les résultats de nos expérimentations sont représentés par la Figure 4.11.

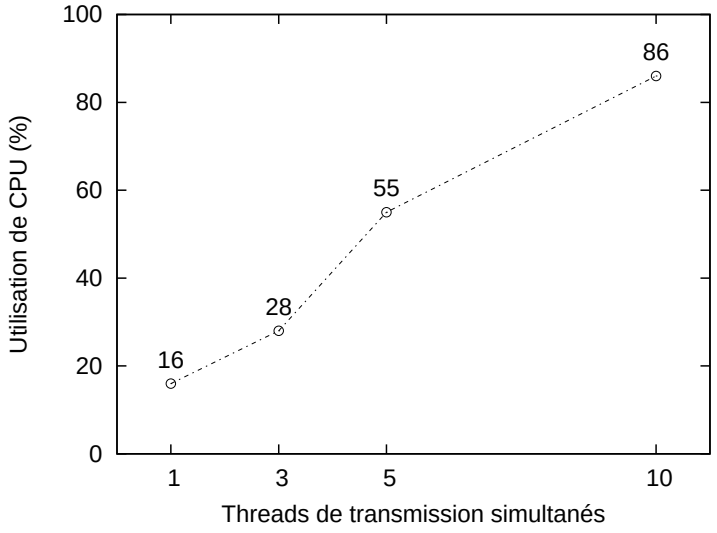


Figure 4.11 – Usage du CPU en fonction du nombre de threads simultanés dans un Streamer.

Cette figure montre que l'utilisation du processeur s'accroît, de manière presque linéaire, avec l'augmentation du nombre de threads gérés simultanément par le Streamer. Toutefois, l'utilisation du processeur devient excessive au-delà d'un certain nombre de threads simultanés. Ce nombre s'élève à 10 dans la configuration de nos tests. Ainsi, il est indispensable pour un usager de limiter le nombre maximal de threads de transmission qui seront accueillis simultanément sur sa machine, de manière à ce qu'elle ne soit submergée par le traitement et la transmission des contenus partagés, et permettant donc à l'usager d'utiliser son matériel confortablement. Notons que si un Streamer est appelé à prendre en charge plus de flux que le matériel sous-jacent ne peut en supporter, les contraintes temps réel inhérentes aux streams risquent de ne plus être respectées.

4.5 Conclusion

Au travers d'une description détaillée de la conception et de la mise en œuvre du prototype JStreaper, nous avons montré qu'il est possible de déployer à petite échelle un système P2P pour le partage de contenus audiovisuels en mode streaming. Dans un tel système, il n'y a pas de serveur dédié au stockage ni à la diffusion des données multimédias, ces tâches étant assurées par les pairs eux-mêmes. Chaque Streaper prend en charge, dans la mesure du possible, le streaming de ses contenus partagés. D'ailleurs, la capacité d'un Streaper à adapter les contenus multimédias à une résolution particulière permet même aux usagers équipés de terminaux légers d'utiliser le système. En outre, notre prototype confère une facilité d'installation à ses utilisateurs. En effet, le processus d'installation de JStreaper se limite à une simple copie du fichier `Streaper.jar` dont l'empreinte mémoire est de seulement 160 Ko. Notons au passage que le Dispatcher a une empreinte mémoire de 8 Ko.

En perspective immédiate, nous envisageons de parfaire le prototype JStreaper en décentralisant le Dispatcher. Il s'agit d'étudier dans un premier temps si l'utilisation des tables de hachage distribuées (DHT) dans un environnement spontané est pertinente. Dans l'affirmative, il serait intéressant de distribuer l'annuaire des contenus partagés entre les Streapers. Cet annuaire est constitué à l'heure actuelle d'un simple fichier XML centralisé sur une machine considérée fiable. Ainsi, JStreaper pourrait être transformé en un système de streaming pur P2P. Il serait alors très intéressant de conduire des séries de tests extensifs afin de mesurer les performances du prototype JStreaper.

Comme nous l'avons vu au chapitre précédent, le comportement volatil des utilisateurs peut compromettre le mode de partage des contenus multimédias au sein de JStreaper. En effet, le départ d'un usager partageant un contenu en cours d'accès par un autre risque de frustrer ce dernier. Pour cette raison, nous pensons qu'il serait judicieux d'inclure les protocoles développés au chapitre précédent à JStreaper. Une telle inclusion permettrait de mettre, dans un futur proche, ces protocoles à l'épreuve et de mesurer leurs performances via des tests en grandeur réelle.

* * *

Conclusion

L'informatique de demain, telle que nous la percevons aujourd'hui, sera distribuée et multimédia. Dans cette optique, le streaming des données audiovisuelles apparaît comme un champ d'application particulièrement prometteur, à condition de disposer des matériels et logiciels susceptibles de répondre aux contraintes très sévères imposées par ce type d'applications : volume de données gigantesque, forte charge transactionnelle et conditions d'exécution temps réel.

Dans cette thèse, nous avons donné des solutions, dont la pertinence a été validée par simulation, à deux problématiques. D'une part, nous avons apporté un nouvel éclairage à la navigation accélérée dans les présentations multimédias, fonctionnalité absente des systèmes de streaming actuels. Notre démarche consiste à fonder cette navigation sur la structure sémantique de ces présentations. D'autre part, concernant la problématique de la disponibilité des données dans un système de streaming P2P dépourvu de serveur, nous avons contribué à lever ce verrou technologique en proposant d'utiliser un cache distribué entre les pairs.

S'articulant autour de ces deux problématiques, les recherches menées dans cette thèse, sont synthétisées dans les paragraphes suivants. Après en avoir présenté le bilan, nous terminons en mettant en relief les perspectives ouvertes par nos travaux.

Synthèse

➤ Dans le chapitre 2, nous nous sommes intéressés à la navigation accélérée dans les présentations multimédias. Cette navigation, telle qu'elle est effectuée aujourd'hui dans les vidéos, ne peut être utilisée dans le cadre de présentations multimédias. En effet, une restitution accélérée peut rendre une présentation inintelligible, l'auditeur peut alors se trouver dans l'impossibilité de percevoir certaines informations (un texte présenté pendant une période insuffisante pour le lire) et/ou d'effectuer certaines interactions contextuelles (bouton permettant d'afficher une animation illustrative). De plus, dans le cas d'un accès distant à une présentation, une accélération de sa restitution provoque une consommation excessive de ressources en termes de bande passante réseau, puissance de traitement et/ou taille de mémoire vive. Afin de répondre à ces problématiques, tant conceptuelles que techniques, nous nous sommes focalisés sur la sémantique même de la navigation accélérée dans une présentation multimédia.

En révisant la sémantique sous-jacente à une telle navigation, nous avons constaté que l'objectif recherché par un usager lorsqu'il active une action d'avance (ou de recul) rapide est d'accéder rapidement à une information souhaitée. Nous avons donc défini une nouvelle sémantique du parcours rapide qui relie la navigation accélérée au sein d'une présentation à son contenu. Au lieu d'une restitution accélérée, le nouveau parcours rapide consiste à passer de thème en thème dans la présentation. La nouvelle navigation accélérée se traduit ainsi par des sauts dans la présentation, d'une idée vers une autre. L'atout de cette sémantique est qu'elle conserve la compréhensibilité d'une présentation lors du parcours rapide de celle-ci.

La concrétisation du nouveau parcours rapide passe par la détection des idées dans une présentation multimédia. Afin d'éviter à l'auteur de marquer les idées intrinsèques à sa présentation, nous avons élaboré une méthode pour détecter ces idées automatiquement. Nous avons utilisé les méta-données de synchronisation entre les objets composant la présentation afin d'y dépister les idées. Pour cela, nous sommes partis des observations suivantes. Le démarrage (ou l'apparition) d'un objet média annonce généralement le début d'une idée. Aussi, l'arrêt (ou la disparition) d'un objet annonce la fin d'une idée et par conséquent déclare le début d'une autre. Ainsi, nous avons utilisé les instants de début et de fin des objets dans une présentation pour y marquer les idées par des points d'interaction. Le nouveau parcours rapide d'une présentation consiste donc à effectuer des sauts vers ces points, la présentation est alors restituée à partir de ces points d'interaction.

Dans le cas d'un accès distant, la restitution de la présentation à partir d'un point d'interaction est mise en pause jusqu'à l'arrivée des données correspondantes. Un temps d'attente est donc subi par l'utilisateur suite à chaque action de navigation accélérée. Ce temps d'attente conditionne la fluidité de la présentation et dégrade par conséquent la qualité du service offert par le système. Afin de réduire ce temps d'attente, nous avons opté pour l'utilisation d'un *cache* où les données sont chargées préalablement à leur restitution. Nous avons élaboré une heuristique de *préchargement*, spécialement adaptée à la nouvelle sémantique du parcours rapide.

Notre politique de gestion de cache, nommée CPS, attribue aux différents blocs de données de la présentation des valeurs reflétant leur pertinence au remplacement ou au préchargement. Les valeurs les plus grandes sont attribuées aux blocs qui suivent le début des idées, puis les valeurs décroissent au fur et à mesure de l'éloignement de chaque début d'idée. Ainsi fondée sur le contenu d'une présentation, la politique CPS s'appuie sur ces valeurs de pertinence pour sélectionner les blocs qui sont candidats au préchargement. On précharge en priorité les blocs ayant des valeurs de pertinence les plus élevées. Une fois rapatriés, ces blocs son préservés dans le cache pour être restitués à l'utilisateur lorsqu'il active une action de parcours rapide. Dans le cas où le cache est plein, les blocs ayant les valeurs de pertinence les plus faibles sont remplacés. Comme le montrent les résultats de nos simulations (cf. Section 2.3.2), cette politique permet de réduire de manière considérable le temps de réponse à une action de navigation accélérée par rapport à LRU-P dont la politique de remplacement est basée sur la date d'accès aux blocs. Le gain apporté par CPS peut atteindre jusqu'à 80% lorsque le seuil de préchargement est judicieusement configuré.

➤ Dans le chapitre 3, nous nous sommes focalisés sur les systèmes de streaming P2P. Nous les avons étudié et identifié leurs lacunes. La disponibilité des données s'avère être un des points-clefs qui doivent être traités de front au sein de tels systèmes. Cette problématique provient principalement du départ imprévisible des pairs. En effet, un tel départ a comme conséquence

directe de compromettre la disponibilité des contenus partagés par ces pairs. Un usager qui accédait à l'un de ces contenus peut alors se trouver dans l'impossibilité de terminer le visionnage du contenu en question. Ce problème est particulièrement critique pour un usager qui est sur le point de terminer le visionnage d'un contenu.

Nous avons modélisé cette problématique afin d'avoir un cadre formel permettant de mieux l'appréhender. Notre avons ensuite élaboré une solution consistant à garantir la disponibilité des données uniquement pour les contenus qui sont en cours d'accès et qui seront vraisemblablement visionnés jusqu'à la fin. Pour ce faire, nous mettons les suffixes de ces contenus dans un cache, tant que l'espace disponible dans ce dernier le permet. Nous avons montré qu'en consacrant les ressources du système P2P pour garantir la disponibilité de ces suffixes, il est possible de réduire les effets de la déconnexion des pairs sur la disponibilité des données, tout en limitant le gaspillage de ces ressources.

Afin de décider s'il est judicieux de consommer des ressources du système P2P pour garantir la disponibilité d'un contenu en cours d'accès, c'est-à-dire de décider si le suffixe de ce contenu est susceptible d'être mis en cache, nous nous sommes appuyés sur un modèle probabiliste. Ce modèle donne la probabilité qu'un utilisateur, qui visionnait un contenu depuis son début jusqu'à une position donnée, continue son visionnage jusqu'à la fin. Un suffixe d'une vidéo est candidat à la mise en cache lorsque la probabilité qu'un pair termine cette vidéo dépasse un certain seuil *Pcache* ($0 \leq Pcache \leq 1$). Dans le cas où la mise en cache d'un suffixe est jugée pertinente, ce modèle sera aussi utilisé pour calculer la taille du suffixe ainsi que l'instant où le processus de mise en cache sera déclenché, en fonction du seuil *Pcache*. Ce seuil détermine donc la taille des suffixes ainsi que le nombre de suffixes à mettre en cache à chaque instant.

Quant à la localisation des suffixes à mettre en cache, nous avons utilisé un cache centralisé dans un premier temps. Par la suite, nous avons étendu notre approche vers une architecture distribuée, c'est-à-dire que chaque pair fournit un petit espace cache de telle sorte que l'agrégation de ces espaces forme un cache virtuel distribué (DVC). Afin de ne pas gaspiller l'espace de stockage du DVC, les données mises en cache n'y sont pas répliquées. Dans l'objectif de distribuer les suffixes à mettre en cache entre les pairs de manière équitable, nous avons élaboré une politique d'allocation qui prend en compte le nombre de canaux de transmission disponibles sur chaque machine. Le suffixe est sauvegardé dans l'espace cache fourni par le pair le moins chargé, en terme de canaux utilisés.

La charge globale de streaming des suffixes est tributaire de deux paramètres. D'une part, la taille du DVC qui varie avec l'arrivée et le départ des utilisateurs au fil du temps et conditionne par conséquent le volume des données que l'on peut mettre en cache. D'autre part, l'activité des utilisateurs connectés qui oscille, elle aussi, avec le temps, induisant une variation dans le nombre de contenus transmis en même temps au sein du système. Face à un comportement aussi dynamique, une stratégie de gestion adaptative s'impose. Nous avons donc mis au point une politique de gestion dynamique du DVC. Notre politique observe le rendement du DVC, en terme de capacité à sauvegarder les suffixes à mettre en cache. Ensuite, elle adapte la valeur attribuée au seuil *Pcache* en conséquence. Dans le cas d'un rendement faible, on augmente la valeur de *Pcache*, et *vice versa*. La nouvelle valeur du seuil conditionne le nombre et la taille des suffixes à mettre en cache, et influe donc sur le volume global des données à mettre en cache. Ce volume dicte à son tour le rendement du DVC. Cette adaptation en boucle fermée est effectuée en temps réel à l'aide d'un système asservi que nous avons conçu à cette fin.

Les résultats de nos simulations (cf. Section 3.4) montrent l'efficacité de notre politique adaptative par rapport aux heuristiques traditionnelles consistant à mettre en cache les suffixes en commençant par les plus petits ou les plus populaires après les avoir triés. Notre politique a un temps d'exécution constant alors que la complexité en temps de ses compétitrices est de l'ordre de $n \log n$. De plus, notre politique enregistre des performances similaires à ses concurrentes traditionnelles, en ce qui concerne le gain apporté par la mise en cache des suffixes sur la disponibilité des données. Notre approche est donc tout à fait adéquate pour être mise en place dans un contexte d'exécution temps réel.

➤ Enfin, au chapitre 4 nous avons souligné le fait que la majorité des systèmes de streaming P2P sont conçus pour un déploiement à grande échelle impliquant, pour chaque contenu dans le système, la présence de plusieurs utilisateurs le partageant. La collaboration entre des pairs lors du streaming d'un contenu à un pair le désirant est une caractéristique notoire de ces systèmes. Au travers de la conception et de la mise en œuvre d'un prototype dénommé JStreaper, nous avons démontré la faisabilité d'un système de streaming P2P où la transmission d'un contenu est prise en charge par un seul pair.

Fondé sur une architecture modulaire et paramétrable, JStreaper est implémenté entièrement en Java, ce qui lui confère la caractéristique d'être portable sur toutes les plates-formes compatibles avec Java. Une autre plus-value de ce prototype est qu'il rend la connexion au système possible même pour les usagers qui sont équipés de terminaux peu puissants, en termes de capacités de traitement et/ou d'affichage. En effet, JStreaper permet d'adapter la qualité des contenus transmis en fonction de la configuration du terminal récepteur.

Bilan

Bien que nous soyons parti d'une feuille blanche, nous disposons aujourd'hui, d'une heuristique de préchargement efficace intégrant la structure sémantique des présentations multimédias au cœur du processus d'optimisation, d'un modèle de cache fondé sur une architecture distribuée et permettant d'allier qualité de service et coopération dans un cadre P2P, et enfin du prototype d'un système de streaming P2P adaptatif : JStreaper. L'ensemble de nos travaux a abouti à six publications dont deux dans des journaux internationaux [1, 2] et quatre articles dans des conférences internationales [3, 4, 5, 6], toutes avec comité de sélection.

L'exploitation des caches du côté client est le point commun à nos deux premières contributions. Cependant, nous avons utilisé ces caches locaux de manières singulières et pour des objectifs différents. Tout en ayant deux principes de fonctionnement différents (statique et paramétrable pour l'une, dynamique et adaptative pour l'autre), nos deux politiques de gestion de cache restent tout de même des heuristiques d'optimisation. Elles reposent sur un pari sur l'avenir et visent à mettre en cache les données auxquelles un accès futur est estimé fort probable.

Perspectives

En perspectives communes à nos politiques de gestion de cache, deux pistes de recherche s'avèrent particulièrement intéressantes :

- Les deux politiques de gestion de cache que nous avons développées tout au long de ce mémoire s'appuient sur des modèles probabilistes. À l'heure actuelle, les fonctions de probabilité que nous avons utilisées sont basées sur des suppositions concernant le comportement d'un utilisateur regardant une vidéo ou une présentation multimédia. Néanmoins, nos modèles sont conçus de manière très générale permettant d'interchanger ces fonctions avec d'autres pour peu qu'elles respectent les contraintes imposées dans ces modèles. Afin de concevoir des fonctions de probabilité reflétant la réalité comportementale d'un usager qui regarde un contenu audiovisuel, il serait intéressant de monter des collaborations avec des sociologues spécialisés. Plusieurs paramètres seraient à prendre en compte, parmi lesquels : (a) durée et thème d'un contenu multimédia, (b) langues maîtrisées par un usager par rapport à un contenu donné, (c) corrélation entre le profil d'un usager et le contenu d'une présentation ou d'une vidéo, autrement dit l'intérêt d'un contenu pour un utilisateur en particulier.

- Par ailleurs, trouver le meilleur jeu de paramètres de nos heuristiques d'optimisation est un challenge qui constitue lui-même un problème d'optimisation. Il s'agit pour la politique CPS du seuil de préchargement β et du paramètre α ; quant à la mise en cache des suffixes, les paramètres sont les bornes inférieure et supérieure ainsi que le pas d'incrémentation (ou de décrémentation) de la variable *Pcache*. Afin de sortir de ce cercle vicieux, nous avons fixé les valeurs de ces différents paramètres de manière expérimentale, suite à de nombreux tests. Bien que les valeurs choisies aient donné de bonnes performances, il serait intéressant de mener un travail théorique pour trouver les valeurs optimales de ces paramètres.

Sur le plan fondamental, la nouvelle sémantique de la navigation accélérée dans les présentations multimédias nous semble particulièrement pertinente et mérite d'être étendue à tous les médias continus, vidéo ou audio. La principale difficulté se situe dans l'absence à priori de structures d'annotations des données audiovisuelles. Dans ce cadre, ainsi que nous l'avons déjà souligné, le standard MPEG-7 est notablement prometteur. Les données d'indexation ajoutées aux contenus multimédias conformément à ce standard peuvent être utilisées afin de dériver les idées intrinsèques à ces contenus, ces idées peuvent alors être parcourues rapidement.

Quant à nos travaux portant sur la disponibilité des données dans les systèmes de streaming P2P, les perspectives se révèlent particulièrement riches et s'orientent selon trois principaux axes :

- Premièrement, l'efficacité de la mise en cache des suffixes est étroitement corrélée avec les ressources (espace cache et threads de transmission) fournies par les pairs en échange de l'utilisation du système. Une continuité logique de notre démarche consiste à examiner des méthodes permettant d'inciter les utilisateurs à mettre davantage de leurs ressources à disposition du système, afin de rendre son utilisation plus fiable en terme de disponibilité de données. Il est envisageable d'utiliser un *système de crédits* permettant d'établir un ordre de priorité entre les suffixes à mettre en cache. Le suffixe prioritaire serait alors celui qui est en

cours d'accès par l'utilisateur disposant de plus de crédits. Les crédits pourraient être accordés à un utilisateur en fonction de ses contributions vis-à-vis du système : des crédits tributaires de la taille de l'espace cache fourni, du nombre de threads de transmission réservés et/ou de la durée pendant laquelle ses ressources ont été mises à disposition du système. Les crédits pourraient également être attachés à un profil d'utilisateur afin de lui permettre de se connecter au système via différents équipements, plus ou moins puissants. L'utilisateur pourrait alors tirer profit de ses crédits accumulés par le passé lorsqu'il se connecte via un terminal léger par exemple.

- En second lieu, notre politique de mise en cache de suffixes dans un cache distribué ne stocke jamais le suffixe d'une vidéo en cours d'accès par un pair dans l'espace cache fourni par ce dernier qui est pourtant le consommateur final de la vidéo. Une alternative pourrait alors consister à stocker le reste de la vidéo, non pas sur une machine distante, mais directement chez son consommateur. Dès que ce dernier atteint une position à partir de laquelle il est fort probable qu'il termine son visionnage, la source augmente son débit de transmission. Cette méthode permet d'assurer la disponibilité de la vidéo en considération si la source reste connectée suffisamment longtemps pour envoyer la totalité de la vidéo au consommateur. Néanmoins, une telle démarche suppose que le consommateur est connecté via un terminal pouvant stocker autant de données, ce qui n'est pas forcément le cas. Après une modélisation concrète de cette méthode, qui est loin d'être triviale, une comparaison avec notre approche constitue sans nul doute un sujet de recherche important. De plus, une combinaison des deux méthodes est une piste de recherche qui mérite d'être creusée.

- En dernier lieu, la mise en cache des suffixes n'est à l'heure actuelle pas tolérante aux pannes. En effet, le suffixe d'une vidéo en cours d'accès est stocké dans l'espace cache d'un pair différent de celui qui la regarde et de celui qui la diffuse. Or, il est possible que le pair sauvegardant le suffixe se déconnecte, pour différentes raisons. Le suffixe est alors perdu et la disponibilité des données affaiblit au sein du système. La réplication en de multiples exemplaires d'un suffixe pourrait résoudre ce problème, mais au détriment de l'espace cache. Ces deux paramètres conflictuels rendent la recherche du nombre optimal des répliques ainsi que de leur localisation une problématique particulièrement séduisante.

Enfin, un autre développement naturel de nos travaux consiste en la fusion des deux politiques de gestion de cache, de façon à permettre le parcours rapide des vidéos, encodées en MPEG-7 par exemple, au sein d'un système de streaming P2P. Une telle fusion nous semble être très prometteuse d'un point de vue recherche et mérite d'être approfondie.

* * *

Bibliographie personnelle

Revues internationales avec comité de sélection

[1] Husam Alustwani, Jacques M. Bahi et Ahmed Mostefaoui. Suffix Caching: an approach to ensure data availability in P2P streaming systems. *International Journal on Internet Protocol Technology, Inderscience*, 3(4):245–256, 2008. Sélectionné parmi les meilleurs papiers de [5].

[2] Husam Alustwani, Jacques M. Bahi et Ahmed Mostefaoui. Improving interaction responsiveness of multimedia presentations. *International Journal of Innovative Computing and Applications, Inderscience*, 1:161–170, 2008. Sélectionné parmi les meilleurs papiers de [6].

Conférences internationales avec comité de sélection

[3] Husam Alustwani, Jacques M. Bahi et Ahmed Mostefaoui. JStreaper : a Java Adaptive P2P Streaming System. Dans *IEEE International Conference on Multimedia Computing and Systems (ICMCS)*. 2009. Taux d'acceptation 40%.

[4] Husam Alustwani, Jacques M. Bahi et Ahmed Mostefaoui. Improving Data Availability in P2P Streaming Systems Using Distributed Virtual Cache. Dans *IEEE International Symposium on Multimedia (ISM)*, p. 384–389. 2008. Taux d'acceptation 24%.

[5] Husam Alustwani, Jacques M. Bahi et Ahmed Mostefaoui. Data Availability in P2P Streaming Systems. Dans *IEEE International Conference on Next Generation Mobile Applications, Services, and Technologies (NGMAST), Workshop on Future Multimedia Networking (FMN)*, p. 531–536. 2008. Taux d'acceptation 17%.

[6] Husam Alustwani, Jacques M. Bahi et Ahmed Mostefaoui. Managing VCR Interactions in Multimedia Presentations. Dans *IEEE International Conference on Digital Information Management (ICDIM)*, p. 179–186. 2006.

Rapport de recherche

[7] H. Alustwani, J. Bahi, A. Mostefaoui et M. Salomon. La Technologie Java : une Solution Stratégique pour les Applications Distribuées Interactives. Rapport technique n° RR2008-11, LIFC - Laboratoire d'Informatique de l'Université de Franche Comté, 2008.

* * *

GLOSSAIRE

AAC Advanced Audio Coding, le successeur du standard MP3.

ADPCM Adaptive Differential Pulse Code Modulation, modulation par impulsions et codage différentiel adaptatif. C'est un algorithme non standardisé de compression de données audio avec pertes.

ADSL Asymmetric Digital Subscriber Line.

API Application Programming Interface, interface de programmation avec une bibliothèque constituée de classes et d'interfaces destinées aux développeurs d'applications afin de facilité leur tâche.

ARPANET Advanced Research Projects Agency Network, le premier réseau à transfert de paquets développé aux États-Unis par la DARPA. Il est le prédécesseur d'Internet.

ASCII American Standard Code for Information Interchange, code américain normalisé pour l'échange d'information.

AWT Abstract Window Toolkit, une bibliothèque graphique pour Java. Ella a été introduite dès les premières versions de Java mais depuis Java 1.2, la bibliothèque de gestion de fenêtre officielle est Swing.

CBR Constant Bit Rate, taux d'échantillonnage fixe.

CD-ROM Compact Disc Read Only Memory, un disque optique utilisé pour stocker des données sous forme numérique, destinées à être lues par un ordinateur.

CDA Compact Disc Audio.

CDN Content Delivery Network, un réseau d'ordinateurs répartis géographiquement et interconnectés via des réseaux longue distance possédant de très hauts débits. Sa raison d'être est d'optimiser la transmission de contenus sur Internet.

CORBA Common Object Request Broker Architecture, une norme conçue par l'OMG pour faciliter le développement et le déploiement de logiciels répartis.

CPS Content-Based Prefetching Strategy, une stratégie de gestion de cache que nous avons développée pour permettre une sorte de navigation accélérée dans les présentations multimédias, basée sur leur contenu.

CPU Central Processing Unit, unité centrale de traitement, elle constitue le composant essentiel d'un ordinateur, son rôle est d'interprèter les instructions et de traiter les données d'un programme.

DHT Distributed Hash Table, une table de hachage distribuée est une technologie permettant l'identification et l'obtention d'une information dans un système réparti.

DNS Domain Name System, un système permettant d'établir une correspondance entre une adresse IP et un nom de domaine et, plus généralement, de trouver une information à partir d'un nom de domaine.

DPSR Dynamic Policy of Segment Replication, politique de réplication dynamique de données.

DSL Digital Subscriber Line, ligne numérique d'abonné. Il renvoie l'ensemble des technologies mises en place pour un transport numérique de l'information sur une ligne de raccordement téléphonique.

DVC Distributed Virtual Cache, cache virtuellement distribué, un espace de stockage formé par l'agrégation de plusieurs espaces fournis par des entités distribuées.

DVD Digital Video Disc ou Digital Versatile Disc, disque vidéo numérique.

EDF Earliest Deadline First.

Ethernet Un protocole de réseau classé dans la couche de liaison du modèle OSI.

FCFS First Come First Served, une méthode de traitement des données placées dans une file d'attente selon leur ordre d'arrivée : premier arrivé premier servi.

FEC Forward Error Correction, code correcteur d'erreur.

FIFO First In- First Out, une méthode de traitement des données : première arrivée, première traitée.

FTP File Transfer Protocol, protocole de transfert de fichiers, c'est un protocole de communication destiné à l'échange informatique de fichiers sur un réseau TCP/IP.

GDS GreedyDual-Size, algorithme de remplacement de contenus qui généralise LRU en prenant en compte la taille de ceux-ci.

GDSF GDS-Frequency, algorithme de remplacement de contenus qui améliore GDS en prenant en compte la fréquence d'accès à ceux-ci.

GIC Generalized Interval Caching, politique d'optimisation de l'utilisation de cache. Elle permet le partage de données entre plusieurs streams accédant au même contenu audiovisuel.

GNU GNU est un système d'exploitation composé exclusivement de logiciels libres. Son nom est un acronyme récursif de « Gnu's Not Unix » qui signifie GNU n'est pas UNIX.

GPL GNU General Public License, licence publique générale GNU, une licence qui fixe les conditions légales de distribution des logiciels libres du projet GNU.

GPRS General Packet Radio Service.

GSS Grouped Sweeping Scheme.

GUI Graphical User Interface, interface graphique. Elle est affichée sur le moniteur d'un ordinateur et grâce à elle un utilisateur peut agir avec différents périphériques d'entrée comme le clavier, la souris, la dictée vocale, etc.

HTTP HyperText Transfer Protocol, protocole de transfert hypertexte. C'est un protocole de communication client-serveur développé pour le World Wide Web (WWW).

IETF Internet Engineering Task Force, un groupe informel, international, ouvert à tout individu, qui participe à l'élaboration de standards pour Internet.

INRIA Institut National de Recherche en Informatique et en Automatique, c'est un organisme public civil de recherche français créé le 3 janvier 1967.

IP Internet Protocol, le protocole de communication fondamental de la suite des protocoles Internet. C'est un protocole de communication de réseau informatique, classé dans la couche réseau du modèle OSI.

ISDN Integrated Services Digital Network, une liaison dont la vitesse peut atteindre 2 Mb/s contre 56 Kb/s pour un modem classique.

ISO International Standardization Organization, un organisme de normalisation international composé de représentants d'organisations nationales de normalisation de 158 pays.

ITU-T International Telecommunications Union-Telecommunications standardization, la plus ancienne organisation internationale technique de coordination entre les États et le secteur privé. Elle est rattachée directement aux Nations Unies depuis 1947. Elle est chargée de la réglementation et de la planification des télécommunications dans le monde.

JAR Java ARchive, une archive utilisée pour distribuer un ensemble de classes Java. Les fichiers JAR sont destinés à être exécutés comme des programmes indépendants, dont une des classes est la classe principale.

JDBC Java DataBase Connectivity, une API fournie avec Java, depuis sa version 1.1, permettant de se connecter à des bases de données.

JMF Java Media Framework, un environnement de gestion de données multimédias.

JVM Java Virtual Machine, machine virtuelle Java.

L/MRP Least/Most Relevant for Presentation, une politique de gestion de cache qui intègre le chargement et le remplacement dans le but de supporter des interactions de type avance rapide/recul rapide dans des contenus audiovisuels.

LFF Largest File First, algorithme de remplacement des documents. Il remplace les documents les plus volumineux en premier.

LFU Least Frequently Used, algorithme de remplacement de données. Il remplace les fragments les moins fréquemment utilisés en premier.

LIFC Laboratoire d'Informatique de l'Université de Franche-Comté, il a été créé en 2000 pour réunir les forces de recherche en informatique dans la région de Franche-Comté.

LRU Least Recently Used, algorithme de remplacement de données. Il remplace les fragments utilisés le moins récemment en premier.

MDC Multiple Description Coding, encodage par description multiples.

MDF Most Demanded First, une méthode de traitement des données placées dans une file d'attente selon leur popularité : l'élément le plus populaire est servi en premier.

MHEG Multimedia and Hypermedia Information Coding Experts Group, un standard de l'ISO pour les présentations multimédias.

MOD Media On Demand, média à la demande.

MP3 MPEG-1 Audio Layer 3, le standard le plus populaire de compression audio, il constitue la 3^{e} couche du standard MPEG-1.

MPEG Moving Pictures Experts Group, groupe d'experts de l'ISO chargé du développement de normes internationales pour la compression, la décompression et le traitement de la vidéo, de l'audio et de leur combinaison, de façon à satisfaire une large gamme d'applications.

MRU Lost Recently Used, algorithme de remplacement de données. Il remplace les fragments utilisés le plus récemment en premier.

Myrinet Un protocole de réseau classé dans la couche de liaison du modèle OSI.

NAS Network Attached Storage, un périphérique de stockage relié à un réseau, sa principale fonction est le stockage de données en un gros volume de manière centralisée pour des clients hétérogènes.

NTSC National Television System Committee, standard de la télévision américaine.

ODP-RM Open Distributed Processing Reference Model, une norme conçue par l'ISO pour faciliter le développement et le déploiement de logiciels répartis.

Ogg Un conteneur libre de données audiovisuelles créé par la fondation Xiph, Ogg vient du mot anglais ogging attribué à une manœuvre tactique du jeu Netrek.

OMG Object Management Group, une association américaine à but non-lucratif créée en 1989 dont l'objectif est de standardiser et de promouvoir le modèle objet sous toutes ses formes.

OS Operating System, système d'exploitation.

OSI Open System Interconnection, modèle d'interconnexion des systèmes en réseau, norme proposée par l'ISO pour les communications entre ordinateurs.

P2P Peer-to-Peer, un modèle de communication dont le fonctionnement n'est pas centralisé dans une relation client-serveur comme c'est habituellement le cas, mais décentralisé entre les différentes machines interconnectées via un réseau, lesquelles sont simultanément clients et serveurs des autres machines et aussi passerelles, en passant les données vers leur(s) destinataire(s).

PAL Phase Alternate Line, standard de la télévision européenne.

PDA Personal Digital Assistant, un ordinateur de poche ou assistant personnel.

RAID Redundant Array of Independent Disks, matrice redondante de disques indépendants.

RLE Run-Length Encoding, encodage par plages, un algorithme de compression de données sans perte.

RMI Remote Method Invocation, une implémentation dite « tout Java » du RPC. C'est une API permettant de manipuler des objets Java instanciés sur une autre JVM que celle de la machine locale.

RPC Remote Procedure Call, appel de procédure à distance, un mode de communication de haut niveau. Il permet d'appeler des fonctions situées sur une machine distante de manière transparante.

RR Round-Robin.

RSVP Resource reSerVation Protocol, protocole de réservation de ressources.

RTC Réseau téléphonique commuté.

RTCP Real Time Control Protocol, protocole de contrôle temps réel.

RTP Real-Time Transport Protocol, protocole pour le transfert temps réel.

RTSP Real Time Streaming Protocol, protocole de streaming temps réel.

SECAM SÉquentiel Couleur À Mémoire, standard de la télévision française.

SF Smallest First, une méthode de traitement des données placées dans une file d'attente selon leur taille : l'élément le plus petit est servi en premier.

SIP Session Initialisation Protocol, un protocole multimédia permettant, par exemple, la voix sur réseau IP (VoIP).

SMIL Synchronized Multimedia Integration Language, une spécification du W3C, basée sur XML permettant l'intégration ainsi que la synchronisation de contenus multimédias divers (images, sons, textes, vidéos, animations, flux de texte) afin de créer des présentations multimédias.

SMP Symmetric Multi-Processors, une architecture qui consiste à multiplier les processeurs au sein d'un ordinateur de manière à en augmenter la puissance de calcul.

Swing La bibliothèque graphique officielle pour Java, depuis sa version 1.2.

TCP Transmission Control Protocol, protocole de contrôle de transmission. C'est un protocole de transport fiable sur un réseau. Il est situé au niveau de la couche transport du modèle OSI.

TVDH TV High Definition, télévision à haute définition.

UDP User Datagram Protocol, protocole de datagramme utilisateur. C'est l'un des principaux protocoles de télécommunication utilisés par Internet. Il fait partie de la couche transport du modèle OSI.

UMTS Universal Mobile Telecommunications System, norme pour les réseaux mobiles de troisième génération.

VBR Variable Bit Rate, taux d'échantillonnage variable.

VLC VideoLAN Client, lecteur multimédia dont le développement est coordonné par des étudiants de l'École Centrale Paris.

VOD Video On Demand, vidéo à la demande.

XML eXtensible Markup Language, un langage de balisage extensible. C'est un standard du W3C permettant l'échange de contenus entre systèmes hétérogènes.

BIBLIOGRAPHIE

[ADHC94] F. Arman, R. Depommier, A. Hsu et M.-Y. Chiu : Content-based browsing of video sequences. *In ACM international conference on Multimedia*, pages 97–103, 1994.

[AGG+07] Siddhartha Annapureddy, Saikat Guha, Christos Gkantsidis, Dinan Gunawardena et Pablo Rodriguez Rodriguez : Is high-quality VOD feasible using P2P swarming ? *In ACM international conference on World Wide Web*, pages 903–912, 2007.

[aka] Akamai Technologies, Internet Bottlenecks : the case for Edge Delivery Services, `http://www.akamai.com`, 2009.

[AKG00] Ramon Arean, Jelena Kovacević et Vivek K. Goyal : Multiple description perceptual audio coding with correlating transforms. *IEEE Transactions on Speech and Audio Processing*, 8(2):140–145, 2000.

[All83] James F. Allen : Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[All08] AllCast : Allcast, 2008. `http://www.allcast.com/`.

[AOG92] David P. Anderson, Yoshitomo Osawa et Ramesh Govindan : A file system for continuous media. *ACM Transactions on Computer Systems*, 10(4):311–337, 1992.

[AR03] Stephen Alstrup et Theis Rauhe : *Introducing Octoshape - a new technology for large-scale streaming over the Internet.* EBU Technical Review, 2003.

[AS95] Sedat Akyürek et Kenneth Salem : Adaptive block rearrangement. *ACM Transactions on Computer Systems*, 13(2):89–121, 1995.

[ASA+96] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Edward A. Fox et Stephen Williams : Removal policies in network caches for World-Wide Web documents. *ACM SIGCOMM Computer Communication Review*, 26(4):293–305, 1996.

[ATW02] John G. Apostolopoulos, Wai T. Tan et Susie J. Wee : Video Streaming : Concepts, Algorithms, and Systems. 2002.

[AW01] J. G. Apostolopoulos et S. J. Wee : Unbalanced multiple description video communication using pathdiversity. *In IEEE International Conference on Image Processing (ICPI)*, volume 1, pages 966–969, 2001.

[AWWT02] John G. Apostolopoulos, Tina Wong, Susie J. Wee et Daniel Tan : On multiple description streaming with content delivery networks. *In IEEE Conference on Computer Communications (INFOCOM)*, 2002.

[AWY96] Charu C. Aggarwal, Joël L. Wolf et Philip S. Yu : On Optimal Batching Policies for Video-on-Demand Storage Servers. *In IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pages 253–258, 1996.

[AWY01] Charu C. Aggarwal, Joel L. Wolf et Philip S. Yu : The Maximum Factor Queue Length Batching Scheme for Video-on-Demand Systems. *IEEE Transactions on Computers*, 50(2):97–110, 2001.

[Bab08] Babelgum : Babelgum, 2008. `http://www.babelgum.com/`.

[BB95] Christoph Bernhardt et Ernst Biersack : The server array : a scalable video server architecture. *In International Workshop on Architecture and Protocols for High Performance Networks : High-Speed Networking for Multimedia Applications*, pages 103–125, 1995.

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang et W. Weiss : *An Architecture for Differentiated Service*. RFC Editor, 1998.

[BBD+96] William J. Bolosky, Joseph S. Barrera, Richard P. Draves, Robert P. Fitzgerald, Garth A. Gibson, Michael B. Jones, Steven P. Levi, Nathan P. Myhrvold et Richard F. Rashid : The tiger video file server. *In ACM international workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, 1996.

[BCV08] Jacques M. Bahi, , Raphaïl Couturier et Philippe Vuillemin : JaceP2P, une infrastructure pair-à-pair basée sur le calcul itératif asynchrone. *Technique et Science Informatiques (TSI)*, 27(3-4):457–485, 2008.

[BDET00] William J. Bolosky, John R. Douceur, David Ely et Marvin Theimer : Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. *In ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 34–43, 2000.

[BdWH+03] Ian Burnett, Rik Van de Walle, Keith Hill, Jan Bormans et Fernando Pereira : MPEG-21 : Goals and Achievements. *IEEE MultiMedia*, 10(4):60–70, 2003.

[BG88] Dina Bitton et Jim Gray : Disk shadowing. *In International Conference on Very Large Data Bases (VDLB)*, pages 331–338, 1988.

[BGH96] Hans Werner Bitzer, Christian Gran et Klaus Hofrichter : MAJA : MHEG applications using Java Applets. *In W3C Workshop on Real Time Multimedia and the World Wide Web*, 1996.

[BGJ+05] Dick Bulterman, Guido Grassel, Jack Jansen, Antti Koivisto, Nabil Layaïda, Thierry Michel, Sjoerd Mullender et Daniel Zucker : Synchronized multimedia integration language (SMIL) 2.1 specification. *W3C Recommendation*, 2005.

[BH98] A. Belloum et L. O. Hertzberger : Dealing with One-Timer-Documents in Web Caching. *In IEEE EUROMICRO Conference*, 1998.

[BHJ+98] Dick C. A. Bulterman, Lynda Hardman, Jack Jansen, K. Sjoerd Mullender et Lloyd Rutledge : GRiNS : a graphical interface for creating and playing SMIL documents. *Computer Networks and ISDN Systems*, 30(1-7):519–529, 1998.

[BJ00] Azer Bestavros et Shudong Jin : Popularity-Aware Greedy Dual-Size Web Proxy Caching Algorithms. *In IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 254–261, 2000.

[BK75] B. T. Bennett et V. J. Kruskal : LRU stack processing. *IBM Journal for Research and Development*, pages 353–357, 1975.

[BK98] Susanne Boll et Wolfgang Klas : ZYX - A Semantic Model for Multimedia Documents and Presentations. *In IFIP TC2/WG2.6 Eighth Working Conference on Database Semantics- Semantic Issues in Multimedia Systems*, pages 189–209, 1998.

[BK04] Abdullah Balamash et Marwan Krunz : An overview of web caching replacement algorithms. *IEEE Communications Surveys & Tutorials*, 6(2):44–56, 2004.

[BKB00] Klaus Breidler, Harald Kosch et László Böszörményi : The parallel video server SESAME-KB. *Distributed and parallel systems : from instruction parallelism to cluster computing*, pages 147–150, 2000.

[BMM+08] Naimul Basher, Aniket Mahanti, Anirban Mahanti, Carey Williamson et Martin Arlitt : A comparative analysis of web and peer-to-peer traffic. *In ACM International Conference on World Wide Web*, pages 287–296, 2008.

[BN84] Andrew D. Birrell et Bruce Jay Nelson : Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):39–59, 1984.

[Bol00] Gregory Alan Bolcer : Magi : An architecture for mobile and disconnected workflow. *IEEE Internet Computing*, 4(3):46–54, 2000.

[BP00] Alice Bonhomme et Loïc Prylli : A Distributed Storage System for a Video-on-Demand Server (Research Note). *In International Euro-Par Conference on Parallel Processing*, pages 1110–1114, 2000.

[BPM+06] Tim Bray, Jean Paoli, C. M. McQueen, Eve Maler, François Yergeau et John Cowan : Extensible Markup Language (XML) 1.1 (Second Edition). *W3C Recommendation*, 2006.

[BPS06] Ashwin Bharambe, Jeffrey Pang et Srinivasan Seshan : Colyseus : a distributed architecture for online multiplayer games. *In 3rd conference on Networked Systems Design & Implementation*, pages 12–12, 2006.

[BT89] Dimitri P. Bertsekas et John N. Tsitsiklis : *Parallel and distributed computation : numerical methods*. Prentice-Hall, Inc., 1989.

[BT96] J. C. Bolot et T. Turletti : Adaptive error control for packet video in the internet. *In IEEE International Conference on Image Processing (ICIP)*, volume 1, pages 25–28, 1996.

[BVZ98] M. Berzsenyi, I. Vajk et H. Zhang : Design and implementation of a video on-demand system. *Computer Networks and ISDN Systems*, 30(16–18):1467–1473, 1998.

[BZ92] M. Cecelia Buchanan et Polle T. Zellweger : Specifying temporal behavior in hypermedia documents. *In ACM conference on Hypertext and Hypermedia*, pages 262–271, 1992.

[CDK+03] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron et Atul Singh : Splitstream : high-bandwidth multicast in cooperative environments. *In ACM symposium on Operating systems principles*, pages 298–313, 2003.

[CDKR02] M. Castro, P. Druschel, A. Kermarrec et A. Rowstron : SCRIBE : A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489–1499, 2002.

[Che01] Milton Chen : Design of a virtual auditorium. *In ACM Multimedia*, pages 19–28, 2001.

[CMM02] Russ Cox, Athicha Muthitacharoen et Robert Morris : Serving dns using a peer-to-peer lookup service. *Lecture Notes in Computer Science*, 2429(1):155–165, 2002.

[CRSZ01] Yang Chu, Sanjay Rao, Srinivasan Seshan et Hui Zhang : Enabling conferencing applications on the internet using an overlay muilticast architecture. *ACM Special Interest Group on Data Communications (SIGCOMM)*, 31(4):55–67, 2001.

[CSWH01] Ian Clarke, Oskar Sandberg, Brandon Wiley et Theodore W. Hong : Freenet : A Distributed Anonymous Information Storage and Retrieval System. *In International workshop on Designing privacy enhancing technologies*, pages 46–66, 2001.

[CT97] Shenze Chen et M. Thapar : A fiber channel-based architecture for internet multimedia server clusters. *In IEEE International Conference on Algorithms and Architectures for Parallel Processing*, pages 437–450, 1997.

[DFM01] Roger Dingledine, Michael J. Freedman et David Molnar : *Peer-to-Peer : Harnessing the Power of Disruptive Technology*. O'Reilly, 2001.

[DKK+01] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris et Ion Stoica : Wide-area cooperative storage with cfs. *In ACM symposium on Operating systems principles*, pages 202–215, 2001.

[DKS95] Asit Dan, Martin Kienzle et Dinkar Sitaram : A dynamic policy of segment replication for load-balancing in video-on-demand servers. *ACM Multimedia Systems*, 3(3):93–103, 1995.

[DKSZ03] Ali Dashti, Seon Ho Kim, Cyrus Shahabi et Roger Zimmermann : *Streaming Media Server Design*. Prentice Hall Professional Technical Reference, 2003.

[DS97] Asit Dan et Dinkar Sitaram : Multimedia Caching Strategies for Heterogeneous Application and Server Environments. *Multimedia Tools Applications*, 4(3):279–312, 1997.

[DSS94] Asit Dan, Dinkar Sitaram et P. Shahabuddin : Scheduling policies for an on-demand video server with batching. *In ACM international conference on Multimedia*, pages 15–23, 1994.

[DSST95] Asit Dan, Perwez Shahabuddin, Dinkar Sitaram et Don Towsley : Channel allocation under batching and VCR control in video-on-demand systems. *Journal of Parallel and Distributed Computing*, 30(2):168–179, 1995.

[FCC03] T. Friedman, R. Caceres et A. Clark, éditeurs. *RTP Control Protocol Extended Reports (RTCP XR)*. RFC Editor, 2003.

[FD95] Craig S. Freedman et David J. DeWitt : The SPIFFI Scalable Video-on-Demand System. *In ACM SIGMOD International Conference on Management of Data*, pages 352–363, 1995.

[FG04] Markus Flierl et Bernd Girod : Video Coding with Motion-Compensated Lifted Wavelet Transforms. *Signal Processing : Image Communication*, 19(7):561–575, 2004.

[FGD96] Jocelyne Farhat-Gissler et Isabelle M. Demeure : Automatic Scheduling of Applications with Temporal QoS Constraints :A Case Study. *In IEEE EUROMICRO Conference on Hardware and Software Design Strategies*, pages 581–, 1996.

[FHPW00] Sally Floyd, Mark Handley, Jitendra Padhye et Jörg Widmer : Equation-based congestion control for unicast applications. *In ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 43–56, 2000.

[FKT01] Ian Foster, Carl Kesselman et Steven Tuecke : The anatomy of the grid : Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[FLdM95] Kazi Farooqui, Luigi Logrippo et Jan de Meer : The ISO reference model for open distributed processing : an introduction. *Computer Networks and ISDN Systems*, 27(8):1215–1229, 1995.

[FLNP04] Bruno D. Fabbro, David Laiymani, Jean M. Nicod et Laurent Philippe : A Data Persistency Approach for the DIET Metacomputing Environment. *In International Conference on Internet Computing*, pages 701–707, 2004.

[FLSA+01] H. Fahmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu et L. Hsu : Proxy Servers for Scalable Interactive Video Support. *IEEE Computer*, 34(9):54–59, 2001.

[G.788] IUT-T Recommendation G.711 : *Pulse code modulation (PCM) of voice frequencies*. International Telecommunication Union, 1988.

[GB00] Jamel Gafsi et Ernst W. Biersack : Modeling and performance comparison of reliability strategies for distributed video servers. *IEEE Transactions on Parallel and Distributed Systems*, 11(4):412–430, 2000.

[GBB+97] Lars Geyer, Michael Baentsch, Lothar Baum, Georg Molter, Steffen Rothkugel et Peter Sturm : MHEG in Java – Integrating a Multimedia Standard into the Web. *In International World Wide Web Conference*, 1997.

[GDS+03] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy et John Zahorjan : Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. *SIGOPS Operating Systems Review*, 37(5):314–329, 2003.

[GFTK06] Jens-Uwe Garbas, Ulrich Fecker, Tobias Troger et Andre Kaup : 4D Scalable Multi-View Video Coding Using Disparity Compensated View Filtering and Motion Compensated Temporal Filtering. pages 54–58, 2006.

[GGM99] Christophe Gransart, Jean-Marc Geib et Philippe Merle : *CORBA : Des concepts à la pratique*. Dunod, 1999. 2e édition.

[GIZ96] Shahram Ghandeharizadeh, Doug Ierardi et Roger Zimmermann : An On-Line Algorithm to Optimize File Layout in a Dynamic Environment. *Information Processing Letters*, 57(2):75–81, 1996.

[GJSB05] James Gosling, Bill Joy, Guy Steele et Gilad Bracha : *The Java Language Specification, Third Edition*. Prentice Hall PTR, 2005.

[GL97] Valérie Gay et Peter Leydekkers : Multimedia in the ODP-RM Standard. *IEEE MultiMedia*, 4(1):68–73, 1997.

[GLS94] W. Gropp, E. Lusk et A. Skjellum : *Using MPI : Portable Parallel Programming With the Message-Passing Interface*. MIT Press, 1994.

[Gon01] Li Gong : Jxta : A network programming environment. *IEEE Internet Computing*, 5(3):88–95, 2001.

[Goy01] V. K. Goyal : *Multiple description coding : compression meets the network*, volume 18. IEEE Signal Processing Magazine, 2001.

[GR91] Christer Grewin et Thomas Ryden : Subjective Assessments on Low Bit-Rate Audio Codecs. *In Audio Engineering Society International Conference*, pages 91–102, 1991.

[Gro08] Peer-to-Peer Working Group, 2008 : `http://p2p.internet2.edu/`.

[GS07] Dirk Grunwald et Douglas Sicker : Measuring the network - service level agreements, service level monitoring, network architecture and network neutrality. *International Journal of Communication*, pages 548–566, 2007.

[GSLH99] B. Girod, K. Stuhlmuller, M. Link et U. Horn : Packet loss resilient internet video streaming. *In SPIE Conference on Visual Communications and Image Processing (VCIP)*, pages 833–844, 1999.

[GVK+95] Jim Gemmell, Harrick M. Vin, Dilip D. Kandlur, P. Venkat Rangan et Lawrence A. Rowe : Multimedia storage servers : A tutorial. *IEEE Computer*, 28(5):40–49, 1995.

[GZS+97] Shahram Ghandeharizadeh, Roger Zimmermann, Weifeng Shi, Reza Rejaie, Doug Ierardi et Ta-Wei Li : Mitra : A scalable continuous media server. *Multimedia Tools Appllications*, 5(1):79–108, 1997.

[H.293] IUT-T Recommendation H.261 : *Video Codec for Audiovisual Services at px64 kbit/s*. International Telecommunication Union, 1993.

[H.297] IUT-T Recommendation H.263 : *Video coding for low bit rate communication*. International Telecommunication Union, 1997.

[HBvR94] Lynda Hardman, Dick C. A. Bulterman et Guido van Rossum : The amsterdam hypermedia model : adding time and context to the dexter model. *Communications of the ACM*, 37(2):50–62, 1994.

[hCRZ00] Yang hua Chu, Sanjay G. Rao et Hui Zhang : A case for end system multicast. *In ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 1–12, 2000.

[hCRZ02] Yang hua Chu Rao, S.G. Seshan et S. Hui Zhang : A case for end system multicast. *IEEE Journal on Selected Areas in Communications*, 20(8):1456– 1471, 2002.

[HGG+04] P. Halvorsen, C. Griwodz, V. Goebel, K. Lund et T. Plagemann J. Walpole : Storage System Support for Continuous-Media Applications, Part 1 : Requirements and Single-Disk Issues. *IEEE Distributed Systems Online*, 5(1), 2004.

[HHB+03] Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu et Bharat Bhargava : PROMISE : peer-to-peer media streaming using CollectCast. *In ACM international conference on Multimedia*, pages 45–54, 2003.

[HKF+06] S. B. Handurukande, A.-M. Kermarrec, F. Le Fessant, L. Massoulié et S. Patarin : Peer sharing behaviour in the edonkey network, and implications for the design of server-less file sharing systems. *SIGOPS Operating Systems Review*, 40(4):359–371, 2006.

[HSE96] Andrew Heybey, Mark Sullivan et Paul England : Calliope : a distributed, scalable multimedia server. *In USENIX Annual Technical Conference (ATEC)*, pages 75–86, 1996.

[HSK+99] James Won-Ki Hong, Young-Mi Shin, Myung-Sup Kim, Jae-Young Kim et Young-Ho Suh : Design and implementation of a distributed multimedia collaborative environment. *Cluster Computing*, 2(1):45–59, 1999.

[Huf52] David Albert Huffman : A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[Hui03] C. Huitema : *Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)*. RFC Editor, 2003.

[HW96] Ernst Hairer et Gerhard Wanner : *Solving Ordinary Differential Equations II : Stiff and Differential-Algebraic Problems*. Springer Verla, 1996.

[ILL07] Alan T. S. Ip, Jiangchuan Liu et Senior John Chi-Shing Lui : Copacc : An architecture of cooperative proxy-client caching system for on-demand media streaming. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 18(1):70–83, 2007.

[JB01a] Shudong Jin et Azer Bestavros : GISMO : a generator of internet streaming media objects and workloads. *SIGMETRICS Performance Evaluation Review*, 29(3):2–10, 2001.

[JB01b] Shudong Jin et Azer Bestavros : Greedydual* web caching algorithm : exploiting the two sources of temporal locality in web request streams. *Computer Communications*, 24(2):174–183, 2001.

[JCCP05] Javier B. Jimenez, Denis Caromel, Alexandre D. Costanz et Jose M. Piquer : Balancing active objects on a peer to peer infrastructure. *In IEEE International Conference on The Chilean Computer Science Society*, page 109, 2005.

[JDXB03] Xuxian Jiang, Yu Dong, Dongyan Xu et Bharat Bhargava : Gnustream : A P2P Media Streaming System Prototype. *In International Conference on Multimedia and Expo (ICME)*, volume 2, pages 325–328, 2003.

[JGJ+00] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek et Jr. James W. O'Toole : Overcast : reliable multicasting with on overlay network. *In Symposium on Operating System Design & Implementation*, pages 14–14, 2000.

[JJS93] N. S. Jayant, J. Johnston et R. Safranek : Signal compression based on models of human perception. *Proceedings of the IEEE*, 81(10):1385–1422, 1993.

[JLR+98] Muriel Jourdan, Nabil Layaïda, Cécile Roisin, Loay Sabry-Ismaïl et Laurent Tardif : Madeus, and authoring environment for interactive multimedia documents. *In ACM international conference on Multimedia*, pages 267–272, 1998.

[jmf] Sun Mircosystems Inc. Java Media Framework API Guide, Novembre 1999.

[Joh88] J. D. Johnston : Transform coding of audio signals using perceptual noise criteria. *IEEE Journal on Selected Areas in Communications*, 6(2):314–323, 1988.

[Joo08] Joost : Joost, 2008. `http://www.joost.com/`.

[JSCB95] Divyesh Jadav, Chutimet Srinilta, Alok Choudhary et P. Bruce Berra : Techniques for scheduling I/O in a high performance multimedia-on-demand server. *Journal of Parallel and Distributed Computing*, 30(2):190–203, 1995.

[JTC93] ISO/IEC JTC1/SC29/WG11 : *MPEG-1 : Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1,5 Mbit/s*. International Organisation For Standardisation, 1993.

[JTC99] ISO/IEC JTC1/SC29/WG11 : *MPEG-2 : Generic Coding of Moving Pictures and Associated Audio Information*. International Organisation For Standardisation, 1999.

[JTC00] ISO/IEC JTC1/SC29/WG11 : *MPEG-4 : Generic Coding of Moving Pictures and Associated Audio Information*. International Organisation For Standardisation, 2000.

[KBC+00] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Chris Wells et Ben Zhao : OceanStore : an architecture for global-scale persistent storage. *SIGARCH Computer Architecture News*, 28(5):190–201, 2000.

[KH00] Changick Kim et Jenq-Neng Hwang : An integrated scheme for object-based video abstraction. *In ACM international conference on Multimedia*, pages 303–311, 2000.

[Kha03] Syed Ali Khayam : The Discrete Cosine Transform (DCT) : Theory and Application. Rapport technique ECE802.602, Wireless and Video Communications Laboratory (WAVES), Michigan State University, 2003.

[KHRR02] Jussi Kangasharju, Felix Hartanto, Martin Reisslein et Keith W. Ross : Distributing layered encoded video through caches. *IEEE Transactions on Computers*, 51(6):622–636, 2002.

[KRAV03] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht et Amin Vahdat : Bullet : high bandwidth data dissemination using an overlay mesh. *ACM SIGOPS Operating Systems Review*, 37(5):282–297, 2003.

[KS95] Michelle Y. Kim et Junehwa Song : Multimedia documents with elastic time. *In ACM international conference on Multimedia*, pages 143–154, 1995.

[LCP+05] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma et Steven Lim : A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[Lee98] Jack Y. B. Lee : Parallel video servers : A tutorial. *IEEE MultiMedia*, 5(2):20–28, 1998.

[Lee05] Jack Y. B. Lee : *Scalable Continuous Media Streaming Systems : Architecture, Design, Analysis and Implementation*. Wiley, 2005.

[LGL08] Yong Liu, Yang Guo et Chao Liang : A Survey on Peer-to-Peer Video Streaming Systems. *Peer-to-Peer Networking and Applications*, 1(1):18–28, 2008.

[LKR06] Jian LIANG, Rakesh KUMAR et Keith W. ROSS : The FastTrack overlay : a measurement study. *International Journal of Computer and Telecommunications Networking*, 50(6):842–858, 2006.

[LL97] Wanjiun LIAO et Victor O. K. LI : The Split and Merge (SAM) Protocol for Interactive Video-on-Demand Systems. *In INFOCOM '97, the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1349–1356, 1997.

[LLQ05] Nabil LAYAÏDA, Tayeb LEMLOUMA et Vincent QUINT : NAC, une architecture pour l'adaptation multim/'edia sur le web. *Technique et Science Informatiques (TSI)*, 24(7):789–813, 2005.

[LOP94] Andrew LAURSEN, Jeffrey OLKIN et Mark PORTER : Oracle media server : providing consumer based interactive access to multimedia data. *ACM SIGMOD international conference on Management of data*, 23(2):470–477, 1994.

[LR04] Nabil LAYAÏDA et Cécile ROISIN : Le temps dans les documents - Langage SMIL. *In Document numéeriques – Gestion de contenu*. Techniques de l'Ingéenieur, 2004.

[Lu04] Honghui LU : Peer-to-Peer Support for Massively Multiplayer Games. *In INFOCOM*, 2004.

[LV94] Thomas D. C. LITTLE et Dinesh VENKATESH : Prospects for interactive video-on-demand. *IEEE MultiMedia*, 1(3):14–24, 1994.

[LW00] Jack Y. B. LEE et P. C. WONG : Performance analysis of a pull-based parallel video server. *IEEE Transactions on Parallel and Distributed Systems*, 11(12):1217–1231, 2000.

[LX04] Jiangchuan LIU et Jianliang XU : Proxy caching for media streaming over the internet. *IEEE Communications Magazine*, 42(8):88–94, 2004.

[LY99] Tim LINDHOLM et Frank YELLIN : *The Java™Virtual Machine Specification, Seconed Edition*. Prentice Hall PTR, 1999.

[LZSG03] Edouard LAMBORAY, Aaron ZOLLINGER, Oliver G. STAADT et Markus GROSS : Interactive Multimedia Streams in Distributed Applications. *Computers & Graphics*, 27:735–745, 2003.

[Mar02] J. M. MARTINEZ : MPEG-7 : Overview of MPEG-7 Description Tools, Part 2. *IEEE MultiMedia*, 9(3):83–93, 2002.

[MBE95] Thomas MEYER-BOUDNIK et Wolfgang EFFELSBERG : MHEG Explained. *IEEE MultiMedia*, 2(1):26–38, 1995.

[MBK+97] S. MCCANNE, E. BREWER, R. KATZ, L. ROWE, E. AMIR, Y. CHAWATHE, A. COOPERSMITH, K. MAYER-PATEL, S. RAMAN, A. SCHUETT, D. SIMPSON, A. SWAN, T. L. TUNG, D. WU et B SMITH : Toward a Common Infrastucture for Multimedia-Networking Middleware. *In International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 39–49, 1997.

[MJV96] Steven MCCANNE, Van JACOBSON et Martin VETTERLI : Receiver-driven layered multicast. *In ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, pages 117–130, 1996.

[MKB02] Ahmed MOSTEFAOUI, Harald KOSCH et Lionel BRUNIE : Semantic Based Prefetching in News-on-Demand Video Servers. *Multimedia Tools and Applications*, 18(2): 159–179, 2002.

[MKK95] Frank Moser, Achim Kraiss et Wolfgang Klas : L/MRP : A Buffer Management Strategy for Interactive Continuous Data Flows in a Multimedia DBMS. *In International Conference on Very Large Data Bases (VLDB)*, pages 275–286, 1995.

[MKL+02] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins et Zhichen Xu : *Peer-to-Peer Computing*. HP Laboratories, 2002.

[MKP02] J. M. Martinez, R. Koenen et F. Pereira : MPEG-7 : the generic Multimedia Content Description Standard, part 1. *IEEE MultiMedia*, 9(2):78–87, 2002.

[MM02] Peter Maymounkov et David Mazières : Kademlia : A peer-to-peer information system based on the XOR metric. *In International Peer-to-Peer Symposium*, pages 53–65, 2002.

[MMGC02] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil et Benjie Chen : Ivy : a read/write peer-to-peer file system. *In 5th Symposium on Operating systems design and implementation*, pages 31–44, 2002.

[MMS00] Dan Minoli, Emma Minoli et Larry Sookchand : ITU-T H.320/H.323. *The telecommunications handbook*, pages 4018–4030, 2000.

[MNO+96] C. Martin, P. Narayanan, B. Ozden, R. Rastogi et A. Silberschatz : The fellini multimedia storage server. *In Multimedia Information Storage and Management*. Kluwer Academic Publishers, 1996.

[MO02] Zhourong Miao et Antonio Ortega : Scalable proxy caching of video under storage constraints. *IEEE Journal on Selected Areas in Communications*, 20(7):1315–1327, 2002.

[Mos04] Ahmed Mostefaoui : Conception et mise en oeuvre de serveurs multimédias. *In Gestion des données multimédias*, pages 141–169. Hermès, 2004.

[MRG07] Nazanin Magharei, Reza Rejaie et Yang Guo : Mesh or multiple-tree : A comparative study of live p2p streaming approaches. *In IEEE International Conference on Computer Communications (INFOCOM)*, pages 1424–1432, 2007.

[MSM97] Matthew Mathis, Jeffrey Semke et Jamshid Mahdavi : The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[MSSK99] Sumedh Mungee, Nagarajan Surendran, Douglas C. Schmidt et Yamuna Krishnamurthy : The design and performance of a corba audio/video streaming service. *In IEEE Annual Hawaii International Conference on System Sciences (HICSS)*, volume 8, page 8043, 1999.

[MW03] Detlev Marpe et Thomas Wiegand : A highly efficient multiplication-free binary arithmetic coder and its application in video coding. *In International Conference on Image Processing (ICIP)*, volume 2, pages 263–266, 2003.

[Nap08] Napster : The napster home page, 2008. `http://free.napster.com`.

[Net00] Groove Networks : Introduction to groove. *In Groove Networks White Paper*, 2000.

[Nie07] Nielson : iTunes Popularity to Surpass RealPlayer. *Bandwidth Report*, 2007. `http://www.websiteoptimization.com/bw/0702/`.

[NKN91] Steven R. Newcomb, Neill A. Kipp et Victoria T. Newcomb : The HyTime : hypermedia time-based document structuring language. *Communications of the ACM*, 34(11):67–83, 1991.

[NKS06] Frederic Dang Ngoc, Joaquín Keller et Gwendal Simon : MAAY : A Decentralized Personalized Search System. *In IEEE International Symposium on Applications and the Internet*, pages 172–179, 2006.

[NLN98] Nicolas Niclausse, Zhen Liu et Philippe Nain : A new efficient caching policy for the World Wide Web. *In Workshop on Internet Server Performance (WISP)*, pages 119–128, 1998.

[NZ02] Thinh Nguyen et Avideh Zakhor : Distributed Video Streaming over Internet. *In ACM/SPIE Conference on Multimedia Computing and Networking*, pages 186–195, 2002.

[OTTH92] Ryuichi Ogawa, Eiichiro Tanaka, Daigo Taguchi et Komei Harada : Design strategies for scenario-based hypermedia : description of its structure, dynamics, and style. *In ACM conference on Hypertext*, pages 71–80, 1992.

[Pan93] Davis Yen Pan : Digital audio compression. *Digital Technical Journal*, 5(2):28–40, 1993.

[PB02] S. Podlipnig et L. Böszörmenyi : Replacement Strategies for Quality Based Video Caching. *In IEEE International Conference on Multimedia and Expo (ICME)*, volume 2, pages 49–52, 2002.

[Pee08a] PeerCast : Peercast, 2008. `http://www.peercast.com/`.

[Pee08b] PeerTV : Peertv, 2008. `http://www.peertv.com/`.

[Per04] Fernando Manuel Bernardo Pereira : Using mpeg standards for multimedia customization. *In Signal Processing : Image Communication*, volume 19, pages 437–456. Elsevier Science B.V., 2004.

[PGK88] David A. Patterson, Garth Gibson et Randy H. Katz : A case for redundant arrays of inexpensive disks (RAID). *In ACM SIGMOD international conference on Management of data*, pages 109–116, 1988.

[PKH+97] C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J. C. Bolot, A. Vega-Garcia et S. Fosse-Parisis : *RTP Payload for Redundant Audio Data.* RFC Editor, 1997.

[PPKB07] Fabio Pianese, Diego Perino, Joaquin Keller et Ernst W. Biersack : PULSE : an adaptive, incentive-based, unstructured P2P live streaming system. *IEEE Transactions on Multimedia, Special Issue on Content Storage and Delivery in Peer-to-Peer Networks*, 9(6), 2007.

[PPL08] PPLive : Pplive, 2008. `http://www.pplive.com/`.

[PPS08] PPStream : Ppstream, 2008. `http://www.ppstream.com/`.

[PW99] Kihong Park et Wei Wang : QoS-Sensitive Transport of Real-Time MPEG Video using Adaptive Forward Error Correction. *In IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, volume 2, page 426, 1999.

[PWCS02] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou et Kunwadee Sripanidkulchai : Distributing streaming media content using cooperative networking. *In International workshop on Network and operating systems support for digital audio and video (NOSSDAV)*, pages 177–186, 2002.

[QS04] Dongyu Qiu et R. Srikant : Modeling and performance analysis of bittorrent-like peer-to-peer networks. *SIGCOMM Computer Communication Review*, 34(4):367–378, 2004.

[RAPP06] Alex Rav-Acha, Yael Pritch et Shmuel Peleg : Making a Long Video Short : Dynamic Video Synopsis. *In IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 435–441, 2006.

[RD90] John T. Robinson et Murthy V. Devarakonda : Data cache management using frequency-based replacement. *In ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 134–142, 1990.

[RD01a] Antony Rowstron et Peter Druschel : Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems. *In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[RD01b] Antony Rowstron et Peter Druschel : Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. *In ACM symposium on Operating systems principles*, pages 188–201, 2001.

[RFH+01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp et Scott Schenker : A scalable content-addressable network. *In Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, 2001.

[RIF02] Matei Ripeanu, Adriana Iamnitchi et Ian Foster : Mapping the gnutella network. *IEEE Internet Computing*, 6(1):50–57, 2002.

[RJMB93] Guido V. Rossum, Jack Jansen, K. S. Mullender et Dick C. A. Bulterman : CMIFed : a presentation environment for portable hypermedia documents. *In ACM international conference on Multimedia*, pages 183–188, 1993.

[RS93] Lawrence A. Rowe et Brian C. Smith : A continuous media player. *In International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 376–386, 1993.

[RSC+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley et E. Schooler : *SIP : Session Initiation Protocol*. RFC Editor, 2002.

[RVT96] S. Rao, H. Vin et A. Tarafdar : Comparative evaluation of server-push and client-pull architectures for multimedia servers. *In ACM international workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 45–48, 1996.

[RW94] A. L. Narasimha Reddy et James C. Wyllie : I/O issues in a multimedia system. *IEEE Computer*, 27(3):69–74, 1994.

[Sar04] Nabil J. Sarhan : Caching and Scheduling in NAD-Based Multimedia Servers. *IEEE Transactions on Parallel and Distributed Systems*, 15(10):921–933, 2004.

[SC01] A. Stephens et P. J. Cordell : SIP and H.323 — Interworking VoIP Networks. *BT Technology Journal*, 19(2):119–127, 2001.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick et V. Jacobson : *RTP : A Transport Protocol for Real-Time Applications*. RFC Editor, 2003.

[Sch01] Rüdiger Schollmeier : A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. *In IEEE International Conference on Peer-to-Peer Computing*, page 101, 2001.

[SD00] Dinkar Sitaram et Asit Dan : *Multimedia servers : applications, environments, and design*. Morgan Kaufmann Publishers Inc., 2000.

[SDLS96] Patrick Sénac, Michel Diaz, Alain Léger et Pierre S. Sannes : Modeling logical and temporal synchronization in hypermedia systems. *IEEE Journal on Selected Areas in Communications*, 14(1):84–103, 1996.

[SGG03] Stefan Saroiu, Krishna P. Gummadi et Steven D. Gribble : Measuring and analyzing the characteristics of napster and gnutella hosts. *Multimedia Systeme*, 9(2):170–184, 2003.

[SGMZ04] Kunwadee Sripanidkulchai, Aditya Ganjam, Bruce Maggs et Hui Zhang : The feasibility of supporting large-scale live streaming applications with dynamic application end-points. *ACM SIGCOMM Computer Communication Review*, 34(4): 107–120, 2004.

[SGRV98] Prashant J. Shenoy, Pawan Goyal, Sriram S. Rao et Harrick M. Vin : Symphony : An integrated multimedia file system. *In SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, pages 124–138, 1998.

[SHT97] Simon Sheu, Kien A. Hua et Wallapak Tavanapong : Chaining : A Generalized Batching Technique for Video-On-Demand Systems. *In IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pages 110–116, 1997.

[Sin01] Munindar P. Singh : Peering at Peer-to-Peer Computing. *IEEE Internet Computing*, 5(1):4–5, 2001.

[SJ98] Pramila Srinivasan et Leah H. Jamieson : High quality audio compression using an adaptive wavelet packet decomposition and psychoacoustic modeling. *IEEE Transactions on Signal Processing*, 46(4):1085–1093, 1998.

[SK00] Douglas C. Schmidt et Fred Kuhns : An overview of the real-time corba specification. *IEEE Computer*, 33(6):56–63, 2000.

[Sky08] Skype : Skype, 2008. `http://www.skype.com/`.

[SLM97] O. Sandsta, S. Langorgen et R. Midtstraum : Video server on an ATM connected cluster of workstations. *In IEEE International Conference of the Chilean Computer Science Society (SCCC)*, page 207, 1997.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek et Hari Balakrishnan : Chord : A scalable peer-to-peer lookup service for internet applications. *In SIGCOMM conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, 2001.

[SML99] John R. Smith, Rakesh Mohan et Chung-Sheng Li : Scalable multimedia delivery for pervasive computing. *In ACM international conference on Multimedia*, volume 1, pages 131–140, 1999.

[SQ07] N. J. Sarhan et B. Qudah : Efficient cost-based scheduling for scalable media streaming. *In SPIE Multimedia Computing and Networking Conference (MMCN)*, pages 15–23, 2007.

[SR01] Mihaela Van Der Schaar et Hayder Radha : A hybrid temporal-snr fine-granular scalability for internet video. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):318–331, 2001.

[SRL98] H. Schulzrinne, A. Rao et R. Lanphier : *Real Time Streaming Protocol (RTSP)*. RFC Editor, 1998.

[SRT99] Subhabrata Sen, Jennifer Rexford et Donald F. Towsley : Proxy prefix caching for multimedia streams. *In IEEE INFOCOM Conference on Computer Communications*, volume 3, pages 1310–1319, 1999.

[SS95] R. Schafer et T. Sikora : Digital video coding standards and their role in video communications. *Proceedings of the IEEE*, 83(6):907–924, 1995.

[SWS03] R. Schafer, T. Wiegan et H. Schwarz : *The Emerging H.264/AVC Standard.* EBU Technical Review, 2003.

[SZFY02] C. Shahabi, R. Zimmermann, K. Fu et S. Yao : Yima : A Second-Generation Continuous Media Server. *IEEE Computer*, 35(6):56–64, 2002.

[TATH95] Yukinobu Taniguchi, Akihito Akutsu, Yoshinobu Tonomura et Hiroshi Hamada : An intuitive and efficient access interface to real-time incoming video based on automatic indexing. *In ACM international conference on Multimedia*, pages 25–33, 1995.

[TBVZ00] A. Terzis, B. Braden, S. Vincent et L. Zhang : *RSVP Diagnostic Messages.* RFC Editor, 2000.

[THD03] Duc A. Tran, Kien Hua et Tai Do : A peer-to-peer architecture for media streaming. *IEEE Special Issue on Advances in Service Overlay Networks (JSAC)*, 20(8):1489–1499, 2003.

[TJW03] G. Tan, H. Jin et S. Wu : Clustered Multimedia Servers : A Survey on the Architectures and Storage Systems. *Annual Review of Scalable Computing*, 5:92–138, 2003.

[TWJX03] Guang Tan, Song Wu, Hai Jin et Feng Xian : A Scalable Parallel Video Server Based on Autonomous Network-attached Storage. *In International Conference on Parallel Computing (PARCO)*, pages 423–430, 2003.

[TZ94] David S. Taubman et Avideh Zakhor : Multirate 3-D subband coding of video. *IEEE Transactions on Image Processing*, 3(5):572–588, 1994.

[TZ01] Wai-Tian Tan et A. Zakhor : Real-time internet video using error resilient scalable compressionand TCP-friendly transport protocol. *IEEE Transactions on Multimedia*, 19(2):119–127, 2001.

[Ufa08] Ufasoft : P2P Messenger, 2008. `http://www.ufasoft.com/p2p/`.

[VI96] S. Viswanathan et T. Imielinski : Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4(4):197–208, 1996.

[VIF06] Aggelos Vlavianos, Marios Iliofotou et Michalis Faloutsos : BiToS : Enhancing BitTorrent for Supporting Streaming Applications. *In IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–6, 2006.

[VL98] N. Vasconcelos et A. Lippman : A Spatiotemporal Motion Model for Video Summarization. *In IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, page 361, 1998.

[VNRS02] Jerome Verbeke, Neelakanth Nadgir, Greg Ruetsch et Ilya Sharapov : Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. *In International Workshop on Grid Computing*, pages 1–12, 2002.

[WA97] Roland P. Wooster et Marc Abrams : Proxy caching that estimates page load delays. *Computer Networks and ISDN Systems*, 29(8-13):977–986, 1997.

[WAF99] Susie J. Wee, John G. Apostolopoulos et Nick Feamster : Field-to-frame transcoding with spatial and temporal downsampling. *In IEEE International Conference on Image Processing (ICIP)*, volume 4, pages 271–275, 1999.

[WCK03] Huahui Wu, Mark Claypool et Robert Kinicki : A model for MPEG with forward error correction and TCP-friendly bandwidth. *In ACM International workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pages 122–130, 2003.

[WCM02] Wushao Wen, S. H. Gary Chan et Biswanath Mukherjee : Token-tray/weighted queuing-time (TT/WQT) : an adaptive batching policy for near video-on-demand system. *Computer Communications*, 25(9):890–904, 2002.

[WHZ+01] D. Wu, Y. Hou, W. Zhu, Y. Zhang et J. Peha : Streaming Video over the Internet : Approaches and Directions. *In IEEE Transactions on Circuits and Systems for Video Technology*, volume 11, pages 282–300, 2001.

[Wir97] Stefan Wirag : Modeling of Adaptive Multimedia Documents. *In International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS)*, pages 420–429, 1997.

[WLZ01] Feng Wu, Shipeng Li et Ya-Qin Zhang : A framework for efficient progressive fine granularity scalablevideo coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(3):332–344, 2001.

[WNC87] Ian H. Witten, Radford M. Neal et John G. Cleary : Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, 1987.

[WSL03] Min-You Wu, Wei Shu et Chow-Sing Lin : Odyssey : a high-performance clustered video server. *Software Practice & Experience*, 33(7):673–700, 2003.

[WWL+01] Qi Wang, Feng Wu, Shipeng Li, Yuzhuo Zhong et Ya-Qin Zhang : Fine-granularity spatially scalable video coding. *In IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 1801–1804, 2001.

[WYW01] Kun-Lung Wu, Philip S. Yu et Joel L. Wolf : Segment-based proxy caching of multimedia streams. *In International conference on World Wide Web*, pages 36–44, 2001.

[YCK93] Philip S. Yu, Ming-Syan Chen et Dilip D. Kandlur : Grouped Sweeping Scheduling for DASD-based Multimedia Storage management. *ACM Multimedia Systems*, 1(3):99–109, 1993.

[YX97] Jin Yu et Yuanyuan Xiang : Hypermedia presentation and authoring system. *Computer Networks and ISDN Systems*, 29(8-13):875–886, 1997.

[YY98] Jin Yu et Jin Yu : A Simple, Intuitive Hypermedia Synchronization Model and its Realization in the Browser/Java Environment. *In Asia Pacific Web Conference*, pages 209–218, 1998.

[ZBHJ97] L. Zhang, S. Berson, S. Herzog et S. Jamin : *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification.* RFC Editor, 1997.

[ZBP04] Angelo Zaia, Dario Bruneo et Antonio Puliafito : A Scalable Grid-based Multimedia Server. *In IEEE International Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE)*, pages 337–342, 2004.

[ZHS+04] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph et John D. Kubiatowicz : Tapestry : A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.

[Zim98] Roger Zimmermann : *Continuous media placement and scheduling in heterogeneous disk storage systems*. Thèse de doctorat, University of Southern California, 1998.

[ZL05] Roger Zimmermann et Leslie Liu : ACTIVE : Adaptive low-latency peer-to-peer streaming. *In SPIE/ACM Multimedia Computing and Networking*, volume 5680, pages 26–37, 2005.

[ZLLY05] Xinyan Zhang, Jiangchuan Liu, Bo Li et Tak-Shing Peter Yum : CoolStreaming/DONet : A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming. *In IEEE Conference on Computer Communications (INFOCOM)*, volume 3, pages 2101–2111, 2005.

[ÖRS96] Banu Özden, Rajeev Rastogi et Avi Silberschatz : Buffer Replacement Algorithms for Multimedia Storage Systems. *In IEEE International Conference on Multimedia Computing and Systems (ICMCS)*, pages 172–180, 1996.

* * *

Résumé

Deux problématiques principales ont guidé les travaux de cette thèse : (a) comment supporter efficacement les interactions (navigation) dans les présentations multimédias streamées ? et (b) comment améliorer la disponibilité des données dans un système de streaming P2P ?

Afin de permettre une navigation rapide au sein d'une présentation multimédia, diffusée en continu, nous avons proposé une approche exploitant les informations disponibles sur son contenu (les objets la constituant). Ces informations permettent, entre autres, de préserver la cohérence sémantique de la présentation lors des interactions utilisateurs. Dans un deuxième temps, nous avons étudié les performances de notre approche en proposant une stratégie de pré-chargement, nommé CPS (Content-Based Prefetching Strategy), qui a permis de réduire considérablement les temps de latence des interactions utilisateurs.

La disponibilité des données dans les système de streaming P2P diffère fondamentalement de celle observée dans les systèmes P2P classiques, dans le sens où les données consommées sont dépendantes du temps. Ainsi, cette problématique se pose en terme de possibilité au pair destinataire (consommateur) de pouvoir terminer « le visionnage » du contenu vidéo dans sa totalité i.e., durant toute la durée de la vidéo. Or, les systèmes P2P spontanés sont essentiellement caractérisés par leur volatilité fréquente, ce qui pose le problème de la disponibilité des pairs sources pendant le streaming. Nous avons étudié cette problématique en mettant en œuvre, dans un premier temps, un mécanisme de cache centralisé permettant de réduire les effets de la volatilité des pairs et en préservant uniquement les suffixes des vidéos en cours d'accès. Dans un deuxième temps, nous avons étendu notre approche vers un cache virtuel distribué. Les résultats des simulations ont montré la pertinence des approches proposées.

Enfin, nous avons décrit la conception et la mise en œuvre d'un prototype qui démontre la faisabilité d'un système de streaming P2P spontané.

Mots-clefs : streaming, présentation multimédia, interactivité, heuristique, préchargement, système de streaming P2P, disponibilité de données, cache distribué, gestion dynamique, adaptation vidéo.

Abstract

The works in this thesis have been guided by two problems : (a) how to efficiently support fast browsing interactions in streamed multimedia presentations ? and (b) how to enhance data availability in pure P2P streaming systems ?

In order to enable quick browsing within streamed multimedia presentations, we proposed an approach that takes full advantage of object multiplicity in a multimedia presentation. Our approach allows, among other features, to preserve the semantic on the presentation, when a fast browsing interaction occurs. In a second time, we studied the performances of our approach through the proposal of a Content-Based Prefetching Strategy, called CPS. Our strategy enables to considerably reduce the new interaction's latency, that is to reduce the response time of a fast browsing action.

Data availability in P2P streaming systems differs fundamentally from that observed in classical systems, in the sense that the used data are time-dependent. Thus, this problem arises in terms of the opportunity for a peer (consumer) to entirely receive a video content, that is to be able to watch the content to its end. However, spontaneous P2P systems are characterised, mainly, by the volatility of the peers. The unpredictable departure of peers poses the problem of the availability of peers that are sources for streaming. We have studied this problem by setting-up, a centralised caching mechanism to reduce the effects of peer's departure and by only replicating the suffixes (last parts) of the videos that are being accessed. In a second step, we extended our approach towards a distributed virtual cache. The simulation results showed the relevance of the proposed approaches.

Finally, we described the design and implementation of a prototype, that demonstrates the feasibility of a spontaneous P2P streaming system.

Keywords: streaming, multimedia presentation, interactivity, heuristic, prefetching, P2P streaming system, data availability, distributed cache, dynamic management, video adaptation.