Rapport de Recherche

Numéro SINGHOFF-03-2002

Synthèse sur la gestion des ressources dans les systèmes multimédias répartis

F. Singhoff * et I. Demeure †

* LIMI/EA2215
Université de Bretagne Occidentale
20, av Le Gorgeu
29285 BREST

†Ecole Nationale Supérieure des
Télécommunications
CNRS URA 820
46, rue Barrault - 75634 Paris Cedex 13, France

EA 2215
LIMI

Département, IUP d'Informatique
U.F.R. Des Sciences et Techniques
Université de Bretagne Occidentale
6, avenue Le Gorgeu
BP 817
29285 BREST cedex
FRANCE

Mars 2002

Résumé

Les applications traitant des données continues telles que l'audio et la vidéo (applications dites "multimédias"), sont soumises à de nombreuses contraintes de QoS. Les mécanismes de gestion de ressources des systèmes d'exploitation sous-jacents doivent offrir des services adaptés à ces contraintes. Dans ce rapport, nous identifions les principales caractéristiques des applications multimédias : nous présentons ensuite les mécanismes de gestion de ressources qui ont été proposés pour ces applications. Ceux-ci sont illustrés au travers de deux applications multimédias dont les besoins sont diamétralement opposés : une application de vidéo-conférence et une application de télévision interactive sur Internet.

Contents

Chapitre 1

Introduction

Par le passé, les applications informatiques manipulaient essentiellement des données textuelles ou graphiques non animées. Aujourd'hui ces données, parfois qualifiées de "discrètes", sont mélangées à des données telles que le son et les images animées. On parle alors d'applications multimédias (littéralement, applications mélangeant plusieurs média). Les flux tels que le son ou l'image animée sont souvent qualifiés de flux de données "continus" [SN95]. En effet, contrairement aux données discrètes dont les éléments sont temporellement indépendants, les éléments de flux de données continus sont liés par des contraintes temporelles. Ainsi, un flux de données audio est constitué d'une suite d'échantillons qui doivent être délivrés régulièrement au périphérique audio. Il existe donc une contrainte temporelle entre la présentation d'un échantillon donné, celle de son prédécesseur et celle de son successeur. Si cette contrainte n'est pas respectée durant la présentation du flux audio, il risque de ne pas être suffisamment intelligible. Un flux de données continus peut donc être défini comme un flux composé d'éléments de données qui doivent être présentés en respectant des contraintes dites de **qualité de service** (QoS, *Quality of Service*) **temporelles** (exemples : délai entre l'affichage de deux images successives, synchronisation voix-lèvres, synchronisation de l'affichage d'objets animés, etc).

La prise en compte des contraintes temporelles intrinsèques aux médias est donc importante. Il existe une large gamme d'applications multimédias, ayant des besoins en ressources et des contraintes variées. Ceci explique l'existence de solutions diamétralement opposées, allant de l'utilisation de systèmes temps réel fortement contraints, à l'exploitation de systèmes temps partagé tel qu'UNIX (cf. extensions temps réel proposées par SVR4 [GC95] et la norme POSIX.4 [Gal95]). Dans ce rapport de synthèse, nous proposons de présenter et d'illustrer cette diversité.

Pour commencer, dans la partie 2, nous décrivons deux exemples d'applications multimédias illustrant la diversité des besoins. A travers ceux-ci, nous définissons les mécanismes de gestion de ressources pour les applications multimédias abordés dans la suite de ce rapport. Les parties 3, 4, 5 et 6 sont dédiées à la présentation de mécanismes concernant respectivement la gestion du processeur, des ressources réseau, des supports de stockage et de la mémoire. Enfin, nous concluons dans la partie 7.

Chapitre 2

Des besoins applicatifs aux systèmes multimédias

Dans cette partie, nous donnons quelques exemples de contraintes de QoS couramment rencontrées dans les applications multimédias. Nous définissons les mécanismes de gestion de ressources employés dans les systèmes multimédias. Puis, à travers deux exemples d'applications multimédias, nous montrons la diversité des besoins présentés par celles-ci.

2.1 Principales contraintes de qualité de service

Il existe une multitude de contraintes de QoS, et les formalismes utilisés pour les spécifier sont tout aussi nombreux. Dans ce rapport nous ne chercherons pas à les énumérer exhaustivement. Nous distinguons trois groupes de contraintes de QoS : les contraintes de QoS spatiales, les contraintes temporelles et les contraintes de fiabilité [TTCM92].

- Les contraintes spatiales portent sur des aspects quantitatifs. L'utilisateur choisit la taille des images, le nombre de couleurs, etc. Ces contraintes seront traduites en débit nécessaire, en taille de tampons et en volume de données.

- Les contraintes de fiabilité spécifient la marge d'erreur acceptable pour le service demandé. Une contrainte de fiabilité peut exprimer un nombre d'images non délivrées, qui sera traduit en un nombre autorisé de paquets perdus ou en un taux d'erreurs sur bits. Elle peut aussi spécifier les pannes temporelles acceptables. A titre d'exemple, un utilisateur peut tolérer une dérive de la synchronisation voix-lèvres sur un ensemble de flux multimédias donné, mais pas sur d'autres.

- Enfin, les contraintes temporelles expriment des relations d'ordre entre plusieurs événements. Selon que la contrainte se réfère à une horloge ou non, une contrainte est qualifiée d'absolue ou de relative [VT97]. Une contrainte absolue date un événement par la valeur d'une horloge alors qu'une contrainte relative ordonne un ensemble d'événements les uns par rapport aux autres.

  Le rythme d'affichage des images d'un film est une contrainte temporelle. Il en est de même pour la synchronisation d'un flux audio et d'un flux vidéo (on parle de synchronisation voix-lèvres). Les contraintes temporelles sur les flux de données continus sont généralement classées en deux grandes familles : les synchronisations "intra-flux" qui

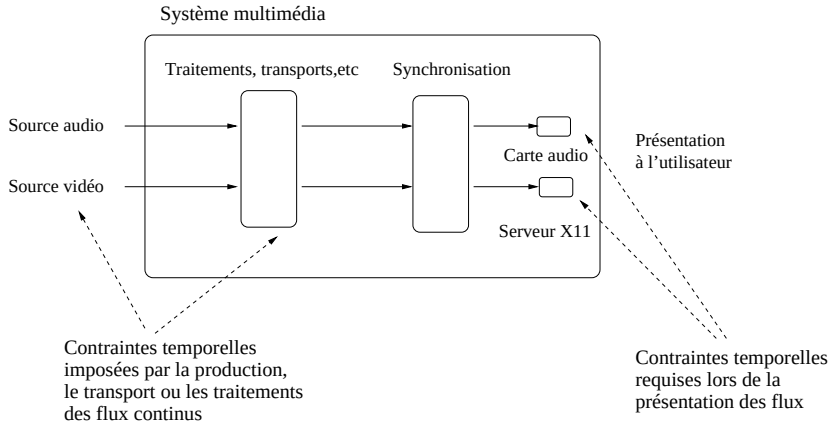

Figure 2.1: Contraintes temporelles présentes dans un système multimédia

spécifient les contraintes temporelles qui existent entre les différents éléments d'un flux multimédia (ce peut être un délai maximal et un délai minimal entre deux images) et les synchronisations "inter-flux" qui décrivent les contraintes temporelles entre plusieurs flux (exemple : synchronisation voix-lèvres). Une contrainte de QoS peut être spécifiée par l'utilisateur ou imposée par un élément du système (cf. figure 2.1).

Mais les contraintes temporelles ne portent pas uniquement sur les flux de données continus. Elles peuvent aussi concerner des événements quelconques du système (exemple : délai maximal entre le début d'un film et la mise à jour d'informations de contrôle sur un écran). On parle alors de contraintes temporelles événementielles [HBD98].

Dans la suite de ce rapport, nous nous intéressons plus particulièrement aux contraintes temporelles de qualité de service.

A chaque contrainte de QoS est associé un niveau de garantie qui définit dans quelle mesure la contrainte sera respectée. On considère généralement trois niveaux de garantie :

- La garantie déterministe qui promet un strict respect de la contrainte de QoS, et ce quelle que soit l'évolution du système.
- La garantie stochastique qui prédit le respect de la contrainte de QoS selon une loi de probabilité donnée.
- La garantie dite "*best effort*" (ou de meilleur effort). Dans ce cas, le système multimédia fait ce qu'il peut afin de respecter la contrainte de QoS sans pour autant offrir de garantie sur le respect de la contrainte.

Une contrainte de QoS peut mettre en jeu plusieurs ressources du système. Exemple : un rythme d'affichage des images se traduit en une contrainte de débit sur un élément réseau, en échéance sur les tâches qui devront délivrer les images, en taille de mémoire, etc. Il faut traduire les contraintes exprimées à haut niveau en contraintes sur les ressources. Ceci nécessite la connaissance, au travers d'un modèle, des ressources du système et des mécanismes de gestion de ces ressources (cf. figure 2.2). Idéalement, c'est l'infrastructure qui offre les moyens de convertir de façon automatique des contraintes de QoS de haut niveau en contraintes de QoS interprétables par les mécanismes de gestion de ressources.

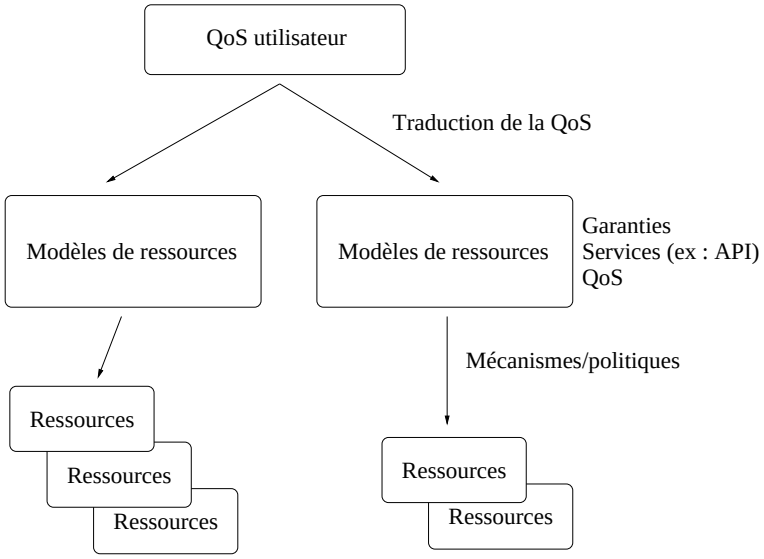

Figure 2.2: Des contraintes à la gestion des ressources du système

Malheureusement, cette tâche est bien souvent réalisée manuellement. En effet, ces traductions successives sont parfois complexes à réaliser du fait, entre autre, de l'interdépendance des ressources d'un système. Or, un système multimédia est composé de nombreux éléments (disque, réseaux, processeurs, carte audio, carte vidéo, codec/decodec audio, carte MPEG, etc). Sans traduction automatique, la tâche des concepteurs d'applications multimédias devient rapidement complexe.

Nous nous concentrons sur les mécanismes de gestion des ressources qui permettent d'assurer le respect des contraintes de QoS. Les formalismes utilisés pour l'expression de la QoS et les mécanismes de traduction de la QoS ne seront pas développés de façon détaillée dans ce rapport. Nous renvoyons le lecteur intéressé par ces aspects vers d'autres publications [VKBG95, CAH98].

2.2 Services offerts par un système multimédia

Définir les contraintes d'un système n'est pas tout : il faut ensuite pouvoir les respecter, entre autre, grâce à des mécanismes de gestion de ressources. Ces mécanismes peuvent être fournis par le système d'exploitation, un intersticiel (ou *middleware*) ou les applications. On peut en distinguer trois catégories : les mécanismes de réservation de ressources, les mécanismes d'allocation de ressources et les mécanismes de supervision.

Les **mécanismes de réservation** de ressources interviennent principalement lors de l'arrivée d'une nouvelle application dans le système. Ce type de service est composé de deux étapes. Une première étape vérifie que toutes les ressources nécessaires à l'application sont disponibles ; c'est l'opération dite de **"contrôle d'admission"**. La deuxième étape est **la réservation** de ressources. La réservation garantit à l'application la disponibilité des ressources voulues durant toute son exécution et ce quel que soit l'évolution du système. La phase de contrôle d'admission est parfois l'occasion d'une **négociation** entre l'application et le système. Dans la négociation, les deux intervenants s'accordent sur une qualité de service commune en fonction des ressources disponibles, des besoins exprimés par l'application et de critères de coût des ressources.

Les **mécanismes d'allocation** décident à chaque instant de la vie du système, à quelle application doit être allouée une ressource donnée.

Enfin, les **mécanismes de supervision** collectent des informations sur l'utilisation des ressources et sur le fonctionnement des applications et du système. Les informations collectées peuvent être exploitées à des fins diverses :

- par les mécanismes de réservation ou d'allocation de ressources, qui les utilisent pour renégocier la qualité de service, limiter ou modifier la consommation en ressources de certaines applications.

- par les applications elles-mêmes. Une fois renseignées sur les ressources disponibles dans le système, les applications adaptent leurs besoins (exemple : choix d'un rythme d'affichage différent, affichage ou non des couleurs, modification de la taille des images affichées, etc).

Cette technique a été utilisée avec succès par [CPS+95] et al. dans la mise en œuvre d'un serveur vidéo à la demande manipulant des fichiers MPEG. Une application de vidéo à la demande offre des services permettant à un ou plusieurs clients de visionner des séquences vidéo/audio choisies parmi une collection stockée sur un support persistant.

Cette application est composée d'un serveur effectuant le transfert des fichiers MPEG par Internet vers un client. Le client effectue la présentation des données à l'utilisateur et transmet périodiquement, vers le serveur, le taux d'occupation du tampon où les images sont stockées avant présentation. Le serveur exploite alors cette information afin d'adapter son rythme d'émission des données. Cette solution permet au serveur de s'adapter aux délais de communication variables sur Internet mais aussi à la capacité de traitement du client, elle aussi éventuellement variable.

Les mécanismes de supervision peuvent donc s'intégrer à des **mécanismes de "rétroaction"** où l'observation des éléments du système conduit celui-ci à adapter le gestion des ressources. L'objectif est que le comportement du système soit le plus proche possible d'une spécification donnée (exemple : qualité de service souhaitée par un utilisateur).

Il existe beaucoup de variantes des mécanismes présentés ci-dessus. Elles reflètent la grande variété des applications multimédias. Le lecteur intéressé par une classification des applications multimédias selon leurs fonctionnalités pourra consulter celle proposée par Hafid et al. [HBD98].

Dans la partie suivante, nous décrivons deux applications qui possèdent des caractéristiques diamétralement opposées. Elles illustrent bien la diversité des besoins en ressources que l'on peut rencontrer dans des systèmes ou applications multimédias.

La première application est une application de vidéo-conférence. La seconde est un service de télévision interactive sur le Web.

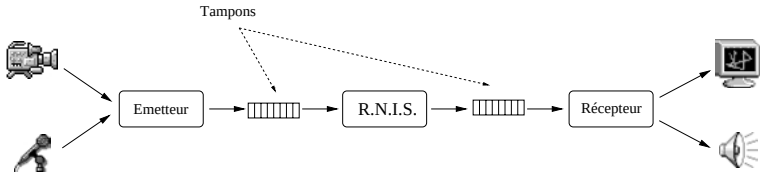

Figure 2.3: Une application de vidéo-conférence

2.3 Une application de vidéo-conférence

Une application de vidéo-conférence a pour objet la transmission de flux audio et vidéo synchronisés d'un émetteur vers un récepteur [GB97]. On souhaite garantir de façon sûre l'acheminement et la restitution des données : aucune variation visible de la synchronisation voix-lèvres ne doit être observée par les deux utilisateurs. Ce type d'application nécessite un contrôle strict de la latence de bout en bout afin de conserver un niveau de réactivité suffisant ; ce qui, entre autre, requiert une gestion fine des tampons et autres zones de stockage de la production (micro, caméra, etc) jusqu'à la présentation (écran, haut-parleur, etc) des informations. Classiquement, on sait que pour des services de transport de la voix, cette latence doit être inférieure à 250 milli-seconde.

Nous supposons que l'émetteur et le récepteur utilisent une plate-forme dédiée pour cette application. Le choix du matériel et du logiciel utilisés font de la plate-forme un environnement d'exécution fortement déterministe (temporellement et logiquement).

Nous supposons aussi que les flux sont compressés selon la norme H.261 [CCI90] et que les informations sont véhiculées par un réseau RNIS (*Réseau Numérique à Intégration de Services*). Le nombre de flux de données continus est fixé à deux : un flux vidéo et un flux audio. Enfin, hormis les instants de démarrage et de terminaison de la vidéo-conférence, nous interdisons à l'utilisateur d'interagir avec l'application.

Du fait de son caractère "statique", il est relativement simple d'évaluer les besoins de cette application multimédia. Regardons, à titre d'exemple, le débit réseau nécessaire au bon fonctionnement de l'application. Tout d'abord, la norme de compression de données H.261 génère un débit constant multiple de 64 Kbits ; or, nous connaissons le nombre de flux qui vont être utilisés dans l'application. Comme le débit d'un film compressé est connu, il est facile de déterminer le nombre de canaux RNIS qui sera demandé par l'application (en divisant le débit généré pour chaque film par 64 Kbits). Ensuite, l'application est, par nature, peu interactive ; les utilisateurs ne peuvent pas modifier les besoins en ressources ou la qualité de service de l'application. En effet, pendant l'exécution de la vidéo-conférence, aucun traitement supplémentaire ne peut être lancé par les utilisateurs (la plate-forme étant dédiée). Le débit réseau de l'application est donc constant. Connaissant précisément le débit nécessaire, notre application de vidéo-conférence utiliserait un mécanisme de réservation de la bande passante. L'utilisation d'un réseau RNIS facilite d'autant plus la mise en œuvre de cette application que ce type de réseau permet la réservation de canaux de communication dont le débit est de 64 Kbits.

Il est possible de suivre le même raisonnement pour les autres ressources de l'application (principalement processeur et mémoire).

2.4 Un service de type "télévision interactive" sur le Web

Voyons maintenant la deuxième application. Celle-ci a pour objet la diffusion grâce au Web de films vers des postes informatiques banalisés. Nous allons, comme pour l'application précédente, poser des hypothèses sur son comportement et ses besoins en ressources.

Contrairement à l'exemple précédent, la télévision interactive offre à l'utilisateur de très nombreuses possibilités d'agir sur l'application. Ainsi lors d'un match de football, nous donnons à l'utilisateur la possibilité de consulter des informations sur les joueurs ou sur des résultats

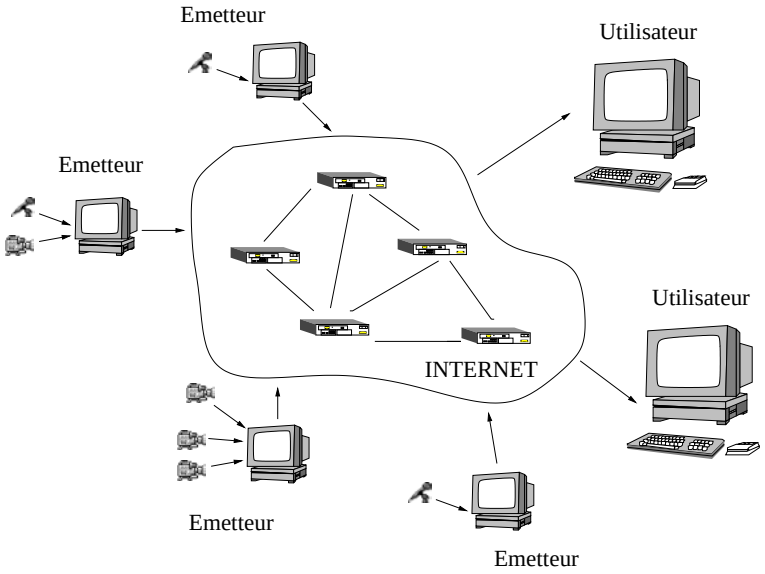

Figure 2.4: Un service de type “télévision interactive” sur le Web

sportifs. Il peut aussi sélectionner le commentaire sportif dans la langue de son choix : si le match se déroule en Allemagne, le téléspectateur français préférera, par exemple, écouter les commentaires en français émis de Paris. Si l'on suppose que les flux audio sont acheminés indépendamment des flux vidéo, cette fonctionnalité pose des problèmes non triviaux de synchronisation voix-lèvres. De même, nous pouvons imaginer que l'utilisateur puisse sélectionner les images de la caméra de son choix, ou puisse regarder le match filmé à partir de plusieurs caméras (caméras qu'il pourra sélectionner au fur et à mesure du déroulement du match).

Contrairement à l'application de vidéo-conférence, l'utilisateur de l'application de télévision interactive influence donc directement les besoins de l'application, en demandant la restitution d'un nombre variable de flux vidéo et audio. Il est donc difficile d'évaluer a priori les besoins en ressources ; évaluation d'autant plus compliquée que la plate-forme utilisée n'est pas complètement déterministe du point de vue temporel. En effet, l'utilisation de postes informatiques banalisés (exemple : PC sous Windows ou Linux) implique une absence de déterminisme temporel que ce soit pour des raisons de matériels (présence de caches mémoire, accès direct à la mémoire, pipe-line, etc), ou des raisons de logiciels (gestion des interruptions et ordonnancement inadéquats, présence de caches disque, etc). Il en est de même pour la ressource réseau ; le réseau Internet ne permettant pas la spécification de contraintes de QoS, et encore moins de les garantir.

Dans ce type de service, même le codage des informations peut être source de problèmes. C'est, par exemple, le cas des chaînes de télévision numérique “à péage” ; ces dernières offrent un service de vidéo à la demande et facturent leurs clients en fonction des films qu'ils ont regardés. Ces réseaux utilisent le standard de compression MPEG-2 [ISO94] pour la diffusion de leur programme [Ric98]. Or, ce standard peut générer des flux vidéo à débits variables (le débit d'un film compressé en MPEG-2 dépend du codeur utilisé et du film lui-même). Il est alors difficile de déterminer a priori de façon précise les besoins en ressources d'un seul flux.

Compte tenu des hypothèses que nous avons faites, on constate que contrairement à l'exemple de la vidéo-conférence, celui de la télévision interactive est beaucoup plus dynamique et que, pour de multiples raisons, ses besoins en ressources sont difficiles à évaluer a priori. De même, il est clair que les besoins en garantie de service de ces deux applications sont différents.

L'utilisateur d'un service de télévision interactive peut éventuellement se satisfaire d'une dégradation temporaire de la qualité de service. Si le système de vidéo-conférence est utilisé pour le suivi d'une opération chirurgicale, une dégradation temporaire de la qualité de service rendue par le système de vidéo-conférence est à exclure. Enfin, les contraintes temporelles exprimées dans ces deux applications peuvent être différentes. Ainsi, si la latence de bout en bout est une contrainte de QoS forte pour la vidéo-conférence, elle l'est moins pour le service de télévision interactif.

2.5 Des classes d'applications multimédias vers les mécanismes de gestion de ressources

A la lumière des exemples précédents, il est clair qu'il serait inadapté d'appliquer les mêmes mécanismes de gestion de ressources à toutes les applications multimédias. Alors que dans le cas de l'application de vidéo-conférence, une grande partie des informations concernant les besoins en ressources peuvent être obtenues hors-ligne (c'est-à-dire avant son exécution), les ressources nécessaires à l'application de télévision interactive sont pour une grande part impossibles à déterminer précisément hors-ligne. Dans ces conditions, une réservation au pire cas conduirait inévitablement à une sur-réservation des ressources, ce qui n'est pas envisageable compte tenu des besoins réels [BCM+96].

Ainsi, il est primordial lorsque l'on construit, évalue ou compare des mécanismes de gestion de ressources, de ne pas oublier les caractéristiques des applications pour lesquelles les mécanismes sont conçus. Dans la suite de ce rapport, nous présentons des exemples de mécanismes que l'on rencontre régulièrement dans la littérature, tout en montrant en quoi ces derniers sont plus ou moins bien adaptés à des applications multimédias dynamiques ou non.

Chapitre 3

L'allocation du processeur

Nous décrivons ici des solutions disponibles aujourd'hui pour l'allocation et la réservation des ressources processeur ; nous verrons dans la suite de ce rapport que des algorithmes similaires sont utilisés pour la gestion d'autres ressources partagées (réseaux, support de masse, etc). Nous nous intéressons uniquement aux systèmes mono-processeurs. Dans un premier temps, nous détaillons les algorithmes et les modèles de tâches conçus pour les applications temps réel qui ont été appliqués dans les systèmes multimédias. Puis, nous examinons quelques modèles de tâches plus spécifiques aux applications multimédias. Enfin, nous terminons en regardant comment les algorithmes d'ordonnancement ont été aménagés pour prendre en compte les aspects adaptatifs de certaines applications multimédias.

3.1 Les algorithmes classiques d'ordonnancement temps réel

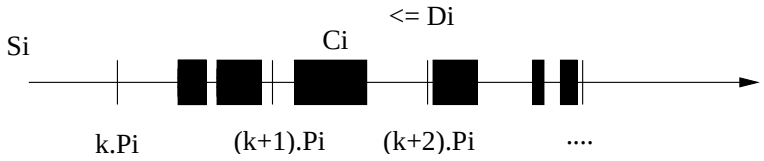

Figure 3.1: Le modèle de tâche périodique de Lui et Layland

Pour spécifier les contraintes temporelles et ordonnancer une application multimédia, on peut utiliser des techniques qui s'adressent aux applications temps réel en général. La littérature sur l'ordonnancement temps réel distingue les tâches répétitives des tâches non répétitives [Leb98]. Les tâches répétitives sont celles qui font l'objet de plusieurs activations successives. A chaque activation, elles effectuent un traitement donné. Le modèle de tâches répétitives le plus répandu est celui de la tâche périodique proposé par Liu et Layland [LL73] (cf. figure 3.1). Dans ce contexte, une tâche périodique i est définie par le quadruplet (S_i, D_i, C_i, P_i) où C_i constitue le temps d'exécution d'une activation de la tâche i (C_i est généralement une borne sur le temps d'exécution), P_i sa période d'activation et D_i son échéance. Enfin, S_i est la date d'arrivée de la tâche dans le système. L'activation k de la tâche i intervient à la date $S_i + P_i(k-1)$. L'échéance D_i est relative aux dates d'activation. Dans le modèle initialement proposé par Liu et Layland, l'échéance d'une tâche était égale à sa période. Une variante de la tâche périodique est la tâche sporadique [Mok83]. Une tâche sporadique est définie par les

mêmes paramètres qu'une tâche périodique mais les activations sont contraintes par un délai minimal et non une période. Enfin, on parle de tâches apériodiques pour désigner des tâches non répétitives définies par le triplet (S_i, D_i, C_i) où S_i est la date d'arrivée de la tâche i dans le système, D_i son échéance et C_i son temps d'exécution.

Pour ordonnancer les tâches périodiques, deux algorithmes ont été proposés par Liu et Layland : RM (*Rate Monotonic*) qui associe une priorité aux tâches de façon statique et EDF (*Earliest Deadline First*) où les priorités sont attribuées dynamiquement.

RM est un algorithme hors ligne : une priorité est affectée à chaque tâche avant l'exécution de l'application. La valeur de cette priorité est inversement proportionnelle à la périodicité de la tâche. Les priorités sont donc fixes. Le processeur est alloué à la tâche prête de plus forte priorité. La majorité des systèmes temps réel et des systèmes UNIX proposant des extensions temps réel disposent d'un ordonnanceur préemptif à priorité fixe. La mise en œuvre d'une application ordonnancée par RM dans ces environnements est donc simple. La réservation de la ressource processeur s'effectue grâce à une analyse hors-ligne du jeu de tâches. Avec le modèle de tâches présenté ci-dessus, il est possible d'utiliser le taux d'occupation du processeur [1]. En effet, le strict respect des échéances des tâches du système peut être garanti si le taux d'occupation respecte une certaine borne. Dans le cas d'un ordonnanceur préemptif, Liu et Layland ont proposé la borne $n(2^{\frac{1}{n}} - 1)$ où n est le nombre de tâches dans le système. Le taux d'occupation du processeur est égal à la somme du rapport entre le temps d'exécution et la périodicité de chaque tâche, autrement dit, si l'inéquation [3.1] est vraie, les contraintes temporelles des tâches seront respectées.

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1) \tag{3.1}$$

Dans le cas d'EDF, l'allocation du processeur est réalisée selon l'échéance des tâches : l'ordonnanceur alloue le processeur à la tâche dont l'échéance est la plus proche. La mise en œuvre d'un tel algorithme à partir d'un ordonnanceur préemptif à priorités fixes est délicate. Dans le cas d'un ordonnanceur préemptif, la réservation de la ressource processeur peut être effectuée comme pour Rate Monotonic grâce à un test d'admission basé sur le taux d'occupation du processeur :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1 \tag{3.2}$$

Les deux algorithmes présentés ci-dessus sont les plus couramment utilisés au sein de la communauté temps réel. Le modèle de tâches qu'ils utilisent est contraignant. Toutefois, de nombreuses extensions ont été proposées. Citons, à titre d'exemple, le support des contraintes de précédence [Bla76, CSB90], la suppression de la contrainte $D_i = P_i$ [LM80, BMR90, Leh90] ou encore la possibilité de partager des ressources entre tâches [SRL90]. Il existe plusieurs travaux de synthèse sur le sujet [CM94, SSNB95, GRS96, HLR97, Leb98, Riv98]. Nous renvoyons donc le lecteur à ces documents.

Les modèles ci-dessus peuvent être utilisés pour des applications multimédias dont le comportement est connu a priori (notre application de vidéo-conférence par exemple). Ainsi Buddhikot et al. utilisent le modèle de Liu et Layland au travers du concept de RTU (*Real Time Upcall*) [BPG96]. Un RTU est une fonction dans l'espace d'adressage de l'application qui peut

[1] D'autres outils existent, en particulier pour des modèles de tâches plus complexes (calcul de temps de réponse, étude de l'ordonnancement généré sur une période de temps, etc).

être exécutée périodiquement par l'ordonnanceur. Le concept de RTU est particulièrement adapté à la mise en œuvre de protocoles de communication applicatifs : en effet, ce concept peut aider à limiter les multiples recopies de données (l'exécution du RTU se faisant dans l'espace d'adressage de l'applicatif). L'ordonnanceur utilise une technique proche de RM. Un contrôle d'admission et une réservation du processeur sont effectués pour chaque RTU nouvellement créé.

Néanmoins, pour un nombre non négligeable d'applications multimédias, il n'est pas possible d'utiliser les mécanismes d'allocation et de réservation décrits ci-dessus sans les aménager. D'abord parce que les modèles de tâches ne sont pas forcément adaptés aux abstractions et contraintes temporelles rencontrées dans les applications multimédias ; ensuite parce qu'il existe des applications multimédias dont le comportement est plus "dynamique". Par comportement dynamique, nous entendons les applications dont les besoins en ressources et/ou les contraintes de QoS peuvent évoluer en cours d'exécution. L'application de télévision interactive sur le Web constitue un bon exemple d'application où ces mécanismes ne sont pas applicables. Nous allons maintenant traiter ces deux points.

3.2 Modèles de tâches pour le multimédia

Les modèles de tâches pour les applications multimédias sont généralement inspirés du modèle de Liu et Layland. Nous allons en présenter deux : les contraintes de distance et les cadences d'activation.

Dans une application multimédia, on s'intéresse souvent à des contraintes autres qu'une échéance relative à une période. Il est courant, par exemple, de spécifier des contraintes de délais minimaux et/ou maximaux entre deux fins de tâche. En effet, un utilisateur peut vouloir spécifier des délais maximum et minimum entre l'affichage de deux images successives. On parle alors de contraintes de "distance" [BZ93, SN95, SGP95, HLH96].

Ce type de contrainte, très fréquente dans les systèmes multimédias, définit de façon indirecte une seconde abstraction tout aussi importante : la notion de gigue [Boc95, GK96]. De façon générale, on appelle gigue une variation sur un délai. En multimédia, pour la résolution des problèmes de dimensionnement par exemple, on s'intéressera plus particulièrement à la "gigue maximale" sur un délai (exemples : délai de communication, délai entre l'affichage de deux images successives, etc). La gigue maximale représente la borne sur la variation du délai. La gigue maximale est définie par la différence entre le délai maximum et le délai minimum. Une contrainte de distance permet donc de spécifier indirectement une gigue maximale.

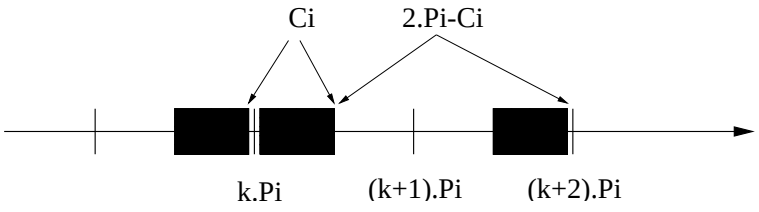

Figure 3.2: Adéquation du modèle périodique au multimédia

Pour définir et ordonnancer ce type de contrainte, plusieurs solutions ont été proposées. Si une tâche périodique est définie par les paramètres (S_i, D_i, C_i, P_i) tel que $D_i = P_i$, le modèle de tâche périodique spécifie uniquement que la terminaison de l'activation $k-1$ doit intervenir

avant le début de la $k$ime période. Pour une tâche i, ceci a pour conséquence de soumettre les terminaisons de ses activations à la contrainte suivante [CCH98] :

$$C_i \leq i_{t+1} - i_t \leq 2 * P_i - Ci$$

où i_t est la date de terminaison de la $t$ime activation de la tâche i (cf. figure 3.2). Avec le modèle de tâche périodique, il est donc possible de spécifier implicitement une contrainte de distance en définissant P_i par :

$$P_i = \frac{\epsilon + C_i}{2}$$

où ϵ constitue un délai maximal entre deux fins d'activation successives. Cette première solution a pour avantage d'utiliser un modèle de tâches et des algorithmes d'ordonnancement bien connus. Toutefois, elle manque singulièrement de souplesse et ne permet pas de spécifier un délai minimum entre deux fins d'activation. Aussi, d'autres techniques ont elles été étudiées pour aménager le modèle périodique de Liu et Layland. Cottet et al. proposent une solution pour supprimer cette variation sur le délai (ou gigue) entre deux fins d'activation successives d'une tâche [CCH98]. La solution consiste à modifier les dates d'exécution au plus tôt ainsi que les échéances. G. Coulson et al. définissent de nouvelles conditions d'ordonnançabilité avec un ordonnanceur EDF sur des tâches qu'ils appellent "isochrones" ou "périodiques avec gigue" [CM97]. Ces modèles de tâches sont des généralisations mieux adaptées aux applications multimédias que ne l'est le modèle de Liu et Layland. Les tâches isochrones sont définies par une échéance égale à leur temps d'exécution. De ce fait, la gigue sur le délai entre deux fins d'activation successives est nulle. Une tâche périodique avec gigue est définie par une échéance différente de la période.

Han et al. proposent un modèle où l'activation des tâches n'est plus déterminée par une période mais par la fin d'exécution de l'activation précédente [HLH96]. Ce nouveau modèle de tâches n'est pas optimal s'il est ordonnancé par EDF. Han et Leboucher [Leb98] élaborent néanmoins un ordonnanceur optimal basé sur EDF pour ce modèle : c'est l'algorithme de la dernière goutte ou LSD (*Last Single Drop*). L'algorithme consiste à retarder la fin d'exécution d'une activation jusqu'à son échéance. Cet algorithme reste toutefois difficile à implanter. Néanmoins, une deuxième solution fût présentée par Han et al. dans [HLH96] : elle manipule un modèle de tâches auquel est ajouté un délai fixe entre la fin d'exécution d'une activation donnée et le début de l'activation suivante.

Enfin, Jeffay et al. proposent un modèle appelé les "cadences d'exécution" [JB95, JG99] (ou RBE pour *Rate Based Execution*). Les tâches d'une application multimédia y sont contraintes par une échéance et sont définies par une loi d'arrivée plus générale que celle initialement décrite par Liu et Layland. Une tâche peut être activée x fois pendant une période de temps y. Aucune hypothèse particulière n'est faite sur la distribution des x activations sur la période de temps y. Le modèle est proche (bien que plus général) du modèle LBAP (*Linear Bounded Arrival Process*) utilisé par Anderson dans Dash [And93]. Les cadences d'exécution ont été implantées dans YARTOS, un système d'exploitation pour les applications de vidéo-conférence [JSS92].

3.3 Prise en compte des applications dynamiques

Dans la partie précédente, nous avons montré que les modèles de tâches temps réel classiques ne sont pas forcément adaptés aux contraintes temporelles exprimées par les applications mul-

timédias.

Nous allons maintenant regarder les solutions qui ont été proposées pour offrir un support satisfaisant aux applications multimédias dynamiques comme celle de notre télévision interactive sur le Web. Nous commençons d'abord par les solutions les plus statiques pour finir par les plus dynamiques.

Les premières solutions que nous pouvons citer sont celles qui ont tenté d'adapter les ordonnanceurs classiques RM ou EDF. Une des techniques les plus connues est celle du serveur sporadique introduite par Lehoczky et al. [SSL89]. L'idée présentée ici est d'intégrer le support de tâches apériodiques dans une application constituée de tâches périodiques et ordonnancées par RM. Dans ces systèmes, l'exécution des tâches apériodiques est déléguée à une tâche périodique. Lehoczky et al. proposent plusieurs stratégies de gestion de cette tâche périodique. Chaque stratégie traite différemment le temps de réponse des tâches apériodiques, le taux d'occupation du processeur, etc.

La solution du serveur sporadique reste toutefois largement insuffisante dans le cadre d'une application telle que celle de la télévision interactive donnée en exemple. Si elle permet, dans une certaine limite, de prendre en compte l'arrivée de nouveaux traitements, elle est incapable de modifier les traitements périodiques déjà existants. Or, être capable de modifier les besoins en ressources des tâches pendant leur exécution est une fonctionnalité primordiale dans ce type d'application.

Le serveur sporadique est, en revanche, une solution intéressante pour l'application de vidéo-conférence si l'on envisage le support de traitements soumis à des contraintes temporelles non strictes (exemple : affichage à la demande d'informations sur les autres utilisateurs, sur l'application, etc). En effet, cette technique garantit le strict respect des contraintes temporelles exprimées par les flux audio et vidéo (grâce aux tâches périodiques) tout en permettant l'exécution de traitements générés durant la vidéo-conférence par l'utilisateur (grâce aux tâches apériodiques).

D'autres solutions pour le support d'applications plus dynamiques ont été proposées. Il existe principalement trois techniques pour adapter les besoins en ressources des tâches en cours d'exécution : renégocier une réservation de ressources, ne pas modifier la réservation mais adapter le comportement de l'application et enfin prévoir hors ligne les évolutions des besoins en ressources de l'application. Nous allons maintenant regarder ces trois classes de solutions.

La première technique peut être illustrée par la solution de Mercer et al., où les tâches sont décrites par une période de réservation et un pourcentage de temps processeur [MST94]. Ce pourcentage est aisément convertible en temps d'exécution d'une activation. Les tâches apériodiques sont dotées d'une période "artificielle" et les algorithmes utilisés sont soit EDF, soit RM. La solution est donc quasi-identique à celle utilisée dans un système temps réel classique. Mais cette fois-ci, aux mécanismes d'allocation et de réservation, les auteurs proposent d'ajouter un mécanisme de rétroaction. Les applications peuvent alors, en fonction des ressources disponibles, renégocier leur réservation de ressources.

Dans les plates-formes multimédias réparties, la technique de la renégociation des ressources est particulièrement répandue ; par exemple, la plate-forme YARTOS [JSS92] réalise un contrôle d'admission à chaque arrivée d'une tâche, puis, permet, le cas échéant, de renégocier les ressources allouées (exemple : si la source n'est plus conforme à la spécification utilisée lors de la réservation).

Toutefois, la renégociation n'est pas l'unique solution pour supporter des applications dynamiques. Hormis son coût important, elle est parfois difficile à utiliser dans une application

(notamment en présence de synchronisations multiples). Une seconde possibilité consiste à demander à l'application d'adapter son comportement sans modifier la réservation effectuée préalablement. L'application effectue alors des traitements différents selon que les ressources sont disponibles ou non. Cette solution est illustrée par Jones et al. [JRR97] ainsi que par Nieh et al. [NL97].

Jones propose que les contraintes de QoS soient directement spécifiées dans le code de l'application [JRR97]. Lors de son démarrage, l'application réserve des unités de temps sur une période donnée. Pendant son exécution, elle délimite les portions de code où doivent s'appliquer les contraintes de QoS. Les contraintes de QoS sont des contraintes d'échéance, de date de début d'exécution et de criticité. Elles ne sont pas spécifiées lors de la réservation du processeur mais au moment où la portion de code doit être exécutée. L'ordonnanceur prédit alors si la contrainte de QoS pourra être respectée (c'est-à-dire si la contrainte de QoS spécifiée reste compatible avec la réservation préalablement effectuée). Le développeur doit prévoir d'exécuter un code différent selon que la contrainte de QoS sera ou non respectée. L'ordonnancement est orienté EDF.

SMART autorise également la spécification de contraintes de QoS, sous la forme d'échéances, sur des portions de code [NL97]. L'application doit fournir une estimation du temps d'exécution pour chaque portion de code. Les informations temporelles sont également intégrées dans le code de l'application. Cette fois, l'ordonnanceur offre un mécanisme d'*upcall* permettant à l'application d'effectuer des traitements lorsqu'une contrainte temporelle ne peut être respectée. L'ordonnanceur de SMART est conçu pour faire coexister des tâches avec et sans contraintes temporelles. Pour ce faire, il utilise deux paramètres par tâche pour effectuer l'allocation de ressources : une notion d'importance et une notion d'échéance. Schématiquement, l'élection d'une tâche par l'ordonnanceur est effectuée en deux étapes : d'abord en construisant l'ensemble des tâches de plus grande importance, puis en choisissant la plus urgente de cet ensemble. Les tâches de même importance partagent de façon équitable la ressource processeur. A cet effet, la notion d'importance est constituée d'une partie statique spécifiée par l'utilisateur et d'une partie dynamique, calculée par l'ordonnanceur.

Enfin, une troisième solution consiste à faire varier les besoins des applications dans un ensemble de valeurs déterminées hors ligne [TTCM92, SM96, BNBH98]. Dans le projet européen Pegasus, Sijben et Mullender proposent d'utiliser cette technique en associant plusieurs niveaux de QoS à chaque application [SM96]. Chaque niveau comprend la liste des tâches et des connexions réseau qui lui sont associées. Une tâche est définie par une période, un temps d'exécution et un critère d'importance. L'ordonnanceur est capable d'exploiter des informations de dépendance entre les tâches. En effet, Pegasus cible tout particulièrement les applications multimédias dont les composants sont agencés en pipeline, architecture très courante dans ce type d'application. Concernant les connexions réseau, les niveaux de QoS précisent le débit nécessaire et la périodicité entre chaque transmission de paquet (ici sur une couche de transport ATM/AAL5). Enfin, les niveaux de QoS précisent aussi la quantité de mémoire nécessaire. Une application fournit donc une liste de niveaux de QoS qu'elle est capable de supporter. La plate-forme Pegasus offre des services de réservation pour la mémoire, le processeur et le réseau. A tout moment, les applications sont donc assurées d'avoir les ressources qu'elles requièrent. Néanmoins, elles ne peuvent décider du niveau de QoS qui va leur être affecté. De plus, le niveau de QoS d'une application peut évoluer au cours de son exécution. En effet, lorsqu'une application entre dans le système, se termine ou modifie ses besoins, les traitements suivants sont déclenchés :

- Un gestionnaire de QoS choisit un niveau de QoS pour chaque application en maximisant

le degré de satisfaction totale du système (grâce au critère d'importance).

- Puis, un ordonnanceur calcule l'ordre d'activation des tâches. Le résultat est stocké dans une liste. L'ordonnanceur est orienté échéance (EDF).

- Enfin, cette liste est transmise au "répartiteur". Le répartiteur exploite la liste des tâches qu'il active séquentiellement. Durant les traitements effectués par l'ordonnanceur et le gestionnaire de QoS, le répartiteur exploite la liste élaborée ultérieurement.

Le mécanisme d'allocation du processeur est donc implanté dans deux entités séparées (l'ordonnanceur et le répartiteur). La solution proposée par Pegasus permet donc de garantir le respect de plusieurs niveaux de contraintes de QoS prédéfinies. Elle est donc plutôt adaptée à l'application de vidéo-conférence.

Nous terminons ce paragraphe par la solution de Brandt et al. [BNBH98]. Brandt et al. n'offrent pas de mécanisme de réservation. Ils n'offrent pas non plus de mécanisme d'allocation du processeur. Comme pour Pegasus, plusieurs niveaux de QoS sont associés à chaque application. Chaque niveau de QoS comprend un pourcentage de processeur ainsi qu'un degré de satisfaction. Un gestionnaire de QoS affecte un niveau de QoS à chaque application en fonction des ressources disponibles et en maximisant un degré de satisfaction. La modification des besoins en ressources est effectuée par les applications qui s'adaptent au niveau de QoS dans lequel le gestionnaire les a placées. La gestion de la ressource processeur est donc réalisée au travers d'un mécanisme de supervision qui permet de détecter les surcharges et les "sous-charges". De part leur souplesse, les mécanismes proposés par Brandt et al. sont plutôt destinés à des applications comme la télévision interactive sur Internet.

Chapitre 4

La gestion des ressources réseau

Dans la partie précédente, nous avons traité le cas de la gestion du processeur, nous regardons maintenant les mécanismes dédiés à la gestion des ressources réseau [Sus00]. Les mécanismes de gestion de ces ressources peuvent être mis en œuvre à plusieurs niveaux dans les systèmes multimédias. On les trouve dans le système d'exploitation ; c'est principalement le cas des couches transport et réseau. Une autre partie du logiciel est embarquée dans les périphériques (exemple : la couche liaison). Enfin, il arrive que les mécanismes soient conçus pour être directement intégrés dans les applications [CT90]. Cette tendance n'est toutefois pas spécifique aux systèmes multimédias. Selon les protocoles utilisés et les services offerts, les mécanismes de réservation bloquent des ressources dans les extrémités et/ou les nœuds du réseau. Ces ressources peuvent être des tampons mémoires, des ressources processeur et de la bande passante. Enfin, les contraintes de QoS généralement rencontrées sont des contraintes temporelles (exemples : délais, gigue, etc.) ou de fiabilité (exemple : taux de perte).

Dans la suite de cette partie, nous présentons tout d'abord des mécanismes bâtis sur des réseaux offrant des services synchrones ou isochrones. Nous nous penchons ensuite sur les réseaux asynchrones [SN95].

Par services asynchrones, nous entendons des services de communication qui n'offrent pas de garantie temporelle sur l'acheminement des données. Ces services sont à opposer aux services synchrones qui garantissent un délai de communication borné et aux services isochrones qui assurent une gigue maximale sur les temps de communication (la gigue est ici la différence entre le temps maximal et le temps minimal de communication ; cf page 12) [Boc95, GK96].

Les services offerts par les réseaux isochrones et synchrones sont typiquement ceux réclamés par l'application de vidéo-conférence donnée en exemple. Dans le cas de l'application de télévision interactive, les mécanismes préconisés sont plutôt ceux proposés pour les réseaux asynchrones.

4.1 Mécanismes rencontrés dans les réseaux synchrones ou isochrones

Les mécanismes rencontrés dans les réseaux synchrones ou isochrones sont généralement des mécanismes de réservation et d'allocation de ressources[1]. Les services de communication offerts dans ce contexte sont orientés connexion. La phase de connexion permet de vérifier la

[1]Dans ces plates-formes, il peut néanmoins exister des services de supervision (exemple : contrôle de flux dans un réseau ATM).

disponibilité des ressources de bout en bout (ou au moins dans les nœuds du réseau), puis de les réserver. Les services d'allocation se chargent par la suite de délivrer un service de communication respectant les garanties promises à la connexion (exemple : les algorithmes d'ordonnancement de paquets dans les routeurs [Zha93]).

Parmi les réseaux offrant des services synchrones, FDDI (*Fiber Distributed Data Interface*) [SP94] est l'un des plus connus. Ce dernier autorise le partage du médium par des communications asynchrones et synchrones. Il a bénéficié, par le passé, d'un intérêt prononcé de la part de la communauté scientifique proposant des solutions pour le support des applications multimédias réparties. Citons à titre d'exemple les travaux d'Anderson [And93], de Zheng [ZS95] ou ceux de Tokuda et al. [TTCM92]. Ces derniers proposent un système de gestion de la QoS où l'utilisateur précise plusieurs niveaux de QoS. Comme pour Pegasus, à chaque niveau est associé un critère d'importance permettant à un gestionnaire de déterminer le niveau de QoS de chaque application, tout en maximisant le degré de satisfaction du système. Ces mécanismes sont implantés dans ARTS [TM89], un système pour les applications temps réel réparties. Le contrôle d'admission et la réservation de ressources sont effectués grâce au protocole CBSRP (*Capacity-Based Session Reservation Protocol*). Les ressources réservées sont la mémoire, la bande passante ainsi que le processeur. Le protocole autorise une renégociation des réservations. Les services de renégociation sont nécessaires puisque la plate-forme peut être amenée à changer le niveau de QoS de certaines applications lors de leur exécution. Tokuda et al. montrent comment, à partir des propriétés temporelles de FDDI, il est possible de traduire les contraintes de QoS utilisateur en contraintes sur la transmission des messages applicatifs. Ils proposent simultanément des règles pour effectuer le contrôle d'admission sur la bande passante synchrone. L'ordonnancement et le contrôle d'admission sur les ressources processeur sont réalisées grâce à RM.

Si FDDI a par le passé fait l'objet de nombreuses propositions pour le support des applications multimédias réparties, il est aujourd'hui remplacé par d'autres technologies. FDDI pose plusieurs problèmes pour la mise en œuvre d'applications multimédias. D'abord, le protocole d'arbitrage du médium de FDDI est basé sur l'échange d'un jeton temporisé. A la configuration du réseau, une valeur cible du temps de passage du jeton est choisie. Hors, en pratique, cette valeur est généralement grande : en effet, plus le jeton passe lentement et plus le réseau est efficace. Malheureusement, ceci augmente aussi la borne sur le délai de communication du service synchrone (au pire cas, cette borne est de deux fois le temps ciblé pour le passage du jeton dans le réseau [SJ87]) ; ce qui implique des besoins en mémoire important si l'on souhaite absorber ce délai de communication (cf. partie 6 page 23). Qui plus est, la bande passante réservée au service synchrone est en pratique peu importante. Enfin, les technologies plus modernes permettent une meilleure intégration de services (utilisation d'un même support physique par des trafics ayant des contraintes différentes).

Le principal successeur de FDDI est ATM [Vet95] (*Asynchronous Transfer Mode*). ATM propose plusieurs services de communication différents [For96]. Dans le cadre d'applications multimédias, deux services sont particulièrement intéressants : le service CBR (*Constant Bit Rate*) et le service VBR-RT (*Variable Bit Rate-Real Time*). Les deux offrent des garanties sur les délais de communication (services isochrones) et sur les débits. Comme leur nom l'indique, le premier est adapté aux trafics à débits constants alors que le deuxième offre un support pour les trafics à débits variables (la réservation s'effectue alors sur des critères de débit crête, débit moyen et taille de rafales). A titre d'exemple, le service CBR est particulièrement bien adapté à un flux de données audio non compressé puisque son débit est constant. Par contre, le service VBR s'applique mieux à des flux de données comme un film compressé par la norme MPEG 2

et qui génère un débit variable.

Malheureusement, les plates-formes ATM qui ont été développées n'offrent pas toujours l'intégralité de ces services. Ainsi, il est peu courant de disposer d'un service VBR. Celui-ci reste difficile à réaliser et à utiliser. Les logiciels distribués par l'EPFL pour le support d'ATM sur Linux offre un service CBR [Alm97]. Ce dernier est accessible soit grâce à une interface de programmation spécifique générant des trames AAL5, soit au travers d'un réseau virtuel Classical-IP [LA98] (dans ce deuxième cas, seul le débit crête peut être spécifié).

Malgré tout, de nombreuses plates-formes multimédias architecturées autour d'un réseau ATM ont été proposées. C'est notamment le cas des travaux de Wray et al. avec la plate-forme Medusa [WG94] ou du système basé sur Chorus proposé par Blair et al. [RCC+94].

Pour terminer, citons un standard émergeant qui offre des services isochrones : le bus IEEE 1394 (ou FireWire) [Wic97]. Ce bus autorise la cohabitation d'un trafic asynchrone et d'un trafic isochrone. Il est particulièrement intéressant dans le cadre d'applications multimédias par sa capacité à interconnecter des machines et périphériques divers (imprimantes, caméras, télévisions, etc). Il offre de plus des débits importants (jusqu'à 400 Mbits/s).

4.2 Mécanismes rencontrés dans les services de communication asynchrones

Nous regardons maintenant le cas des services de communication dit asynchrones. De façon générale, il est difficile dans ces réseaux d'utiliser des mécanismes de réservation de ressources : on a donc plus couramment recours à des mécanismes de supervision. Le principe est d'observer l'état du réseau afin d'adapter le comportement de l'application. On parle de mécanismes de rétroaction. Par exemple, l'observation par le récepteur du taux de pertes des paquets, permet, dans un réseau à commutation de paquets, de détecter une éventuelle congestion [BDS96].

Ces mécanismes sont particulièrement bien adaptés à un protocole comme IP. Ils sont donc largement répandus dans les applications multimédias utilisées sur Internet. Ce principe a d'ailleurs été appliqué avec succès sur des applications multimédias telles que le logiciel IVS de l'INRIA qui offrent des services de vidéo-conférence sur Internet [BTW94, Dio95] ou encore aux décodeurs MPEG répartis de Cen et al. [CPS+95].

Dans le cadre d'Internet, ces mécanismes ont fait l'objet d'une standardisation par l'IETF (*Internet Engineering Task Force*) au travers du protocole RTP [SCFJ96] (*Real Time Protocol*). Ce protocole est très utilisé ; on pourra citer, par exemple, le projet FastWeb [FSVW96], l'outil Vic [Can94], etc. RTP est un protocole utilisateur appliquant le modèle ALF (*Application Level Framing*) [CT90]. Il est généralement implanté au dessus d'UDP. Il permet d'envoyer des paquets de données auxquels sont associées des estampilles temporelles. Ces estampilles permettent alors d'effectuer des synchronisations chez le récepteur (exemple : synchronisation voix-lèvres). RTP est couplé à un protocole de contrôle : le protocole RTCP (*Real Time Control Protocol*). Ce dernier offre des services de supervision mais il n'existe pas, a priori, de liste prédéterminée des paramètres observés. Bien qu'un RFC existe, la philosophie de RTP/RTCP consiste plutôt à estimer les paramètres adaptés à l'application. C'est donc un protocole dont les services offerts dépendent fortement des besoins de l'application. Avec RTCP, il est donc possible, par exemple, d'estimer un taux de perte des paquets, un délai de bout en bout, une gigue inter-arrivée des paquets, etc.

Les protocoles RTP/RTCP, qui ne s'occupent que du transfert des données utilisateurs et des données de contrôle, peuvent être complétés par des protocoles applicatifs comme RTSP

[SRL98]. Ce dernier décrit les interactions possibles entre les serveurs multimédias et les clients souhaitant accéder à des flux de données continus (audio, vidéo). Les interactions proposées par ce protocole intègrent, entre autre, les opérations classiquement offertes par des outils comme les magnétoscopes (lecture accélérée, interruption de la présentation, lecture arrière, etc).

Le protocole RTP n'exige pas que le réseau sous-jacent offre des garanties sur les services de communication (en termes de délai et de débit). Toutefois, il existe des propositions de protocoles et d'architectures émanant de l'IETF dont l'objectif est d'offrir des garanties sur les délais et la bande passante. Ces propositions sont actuellement étudiées par deux groupes de travail : le groupe *intserv* et le groupe $diffserv$. La solution technique proposée par le groupe *intserv* est basée sur une gestion de bout en bout de la QoS et ce pour chaque flux de données [SPG97, Wro97]. On retrouve ici les mécanismes couramment rencontrés dans les réseaux déterministes (contrôle d'admission, lissage du trafic, supervision, etc). La solution est orientée connexion. La réservation de ressources est implantée grâce à un protocole de signalisation, le protocole RSVP [Bra97a, Bra97b] (*ReSerVation Protocol*). Ce dernier conserve dans les routeurs du réseau les informations de QoS et les besoins en ressources pour chaque connexion durant toute leur durée de vie.

Cette première solution pose toutefois des problèmes de déploiement dans le cadre d'un réseau à large échelle tel qu'Internet. En effet, la quantité d'informations véhiculée par le protocole de signalisation ainsi que celle maintenue dans chaque routeur du réseau est incompatible avec le nombre potentiel de connexions. L'approche *intserv* risque donc d'être limitée à des réseaux privés de faible envergure.

Pour contourner ce problème de mise à l'échelle, une alternative est actuellement proposée par le groupe $diffserv$ [BBC+98]. Celle-ci consiste en une agrégation du trafic par classe de QoS. Un nombre fini de classes de trafic est défini statiquement dans le réseau. Lorsqu'une nouvelle connexion est créée, le réseau affecte alors la connexion à une classe de trafic donne. Lors de cette phase d'agrégation, la disponibilité des ressources pour la nouvelle connexion est vérifiée. Par la suite, les paquets appartenant à celle-ci sont labélisés par l'identifiant de la classe de QoS. Cette solution diminue de façon importante les informations maintenues dans les routeurs du réseaux et permet d'espérer une signalisation allégée. Enfin, la phase de contrôle d'admission en est considérablement simplifiée.

Chapitre 5

Les supports de stockage

Nous explorons maintenant les principales solutions techniques existant pour la gestion des supports de stockage permanent. Les mécanismes présentés sont essentiellement des mécanismes de réservation et d'allocation de ressources. Les lecteurs souhaitant obtenir des informations détaillées sur les techniques que nous allons aborder ici, peuvent se référer à l'article de Gemmell et al [GVK+95].

Nous nous plaçons uniquement dans le cadre d'applications dites "de vidéo à la demande" [Gaf99, KBB00]. Les contraintes de QoS sur les support de stockage rencontrées dans ces applications sont essentiellement des contraintes sur la disponibilité des données (tolérance aux pannes), des contraintes de débit (nombre de flux qu'un serveur est capable de délivrer) et des contraintes temporelles (temps de réponse pour lire/écrire des informations depuis/vers une unité de stockage, gigue maximale entre deux lectures successives sur flux, etc). L'application de télévision interactive présentée dans la partie 2.4 exprime des contraintes de QoS similaires.

Pour les contraintes de débit et de fiabilité, les solutions proposées sont principalement architecturales. Ainsi, le débit d'un serveur de vidéo à la demande peut être grandement amélioré par l'utilisation d'un système de stockage hiérarchique (cache mémoire pour les flux les plus demandés, disques magnétiques pour des flux accédés régulièrement et bandes pour les flux utilisés rarement) [BR96]. De même, grâce à leurs mécanismes de redondance, l'utilisation de disques RAID [PGK88], permet d'augmenter sensiblement la fiabilité du support de stockage et éventuellement d'en augmenter le débit (technique de "*striping*").

Pour les contraintes temporelles, il s'agit de techniques d'ordonnancement des requêtes de lecture/écriture ainsi que de choix du placement des données sur le support physique [NRW94, SN95].

Commençons par rappeler la structure d'un disque et ses implications en terme de comportement temporel. Un disque est constitué d'un ensemble de plateaux reliés par un axe central. Chaque plateau contient des pistes concentriques, elles mêmes subdivisées en secteur. Un bras comportant une tête de lecture est associé à chaque plateau. Les bras se déplacent ensemble d'une piste à une autre. L'ensemble des secteurs ainsi accédés sont appelés cylindre.

Le temps de lecture/écriture d'un cylindre comprend deux parties : le temps d'accès au cylindre et le temps de lecture/écriture des informations vers/depuis le support physique. Le temps d'accès à un cylindre se compose du temps de déplacement du bras sur la piste associée au cylindre et du temps de rotation nécessaire afin que le cylindre à accéder passe sous la tête de lecture. Comparativement, et malgré les nombreux progrès technologiques réalisés, le temps d'accès reste grand par rapport au temps pour lire/écrire les données depuis/vers le support physique ; C'est pourquoi plusieurs procédés ont été proposés afin de minimiser ce

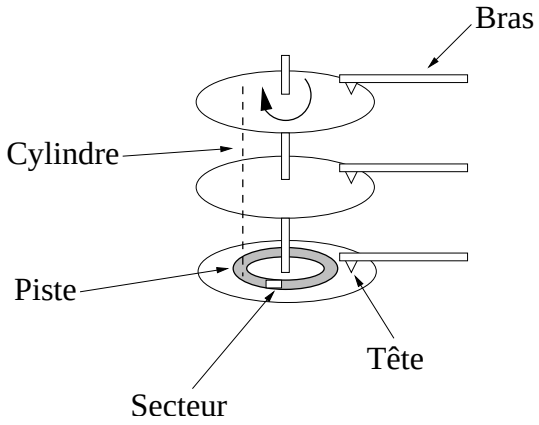

Figure 5.1: Structure d'un disque

temps d'accès.

Une première approche consiste à placer toutes les données de façon contiguë. Le placement contigu peut être complété par l'utilisation d'un cache : le fichier est alors lu en séquence avant que le client en demande la lecture, limitant ainsi le temps d'accès. Cette technique est particulièrement efficace dans le cas de flux multimédias où les données sont presque toujours accédées séquentiellement. Toutefois, le placement contigu des données pose problème lorsqu'il s'agit d'accès en mise à jour. Il implique alors une fragmentation de l'espace disque disponible ainsi qu'un coût élevé lors de l'écriture des données.

Une deuxième approche consiste à ordonner les requêtes de lecture/écriture selon leur position vis-à-vis du bras de lecture [NRW94, SN95]. Il existe principalement deux algorithmes rencontrés : SSTF et SCAN. SSTF (*Shortest Seek Time First*) consiste à servir d'abord les requêtes dont le temps d'accès est le plus court. L'algorithme SCAN parcourt séquentiellement chaque cylindre, et, pour chaque cylindre visité, sert les requêtes qui le concerne. Il s'agit donc de balayer le disque de façon répétitif. Appliqué au multimédia, ces algorithmes ont pour principal défaut le fait qu'ils ne prennent pas en compte les contraintes temporelles associées aux requêtes. C'est pourquoi des solutions hybrides utilisant EDF ont été proposées. Ainsi, comme SCAN, l'algorithme SCAN-EDF effectue un balayage du disque tout en répondant aux différentes requêtes d'un cylindre selon leur échéance.

Bien entendu, comme pour les algorithmes d'ordonnancement que nous avons pu présenter pour les ressources processeur et réseau, le choix d'un algorithme particulier implique des temps de réponse particuliers. Dans le contexte des ressources disque, ceci se traduit en besoin d'espace mémoire (tampon) afin de compenser la gigue possible entre deux opérations d'entrée/sortie successives sur un flux de données continus.

Chapitre 6

Gestion de la mémoire

Nous terminons ce rapport par l'étude des mécanismes utilisés pour la gestion de la mémoire. Comparativement aux ressources réseau et processeur, la gestion de la mémoire a fait l'objet de peu d'attention de la part de la communauté scientifique qui travaille sur les applications multimédias. Celles-ci sont pourtant très gourmandes en mémoire. Les besoins en mémoire interviennent pendant les phases de traitement des données (exemple : décompression d'une image), ou lors des transferts de données depuis ou vers des périphériques (tampons de lecture des disques [GC92, RV93], tampon utilisés lors de transfert réseaux, etc).

De ce fait, les rares équipes qui ont proposé des mécanismes de réservation et d'allocation mémoire spécifiques aux applications multimédias ont essentiellement traité les problèmes de dimensionnement de la mémoire nécessaire au respect d'une contrainte de QoS.

C'est notamment le cas du projet Tenet où Ferrari et al. ont proposé des solutions pour calculer la quantité de mémoire nécessaire au respect de contraintes de QoS déterministes et probabilistes (taux de pertes de paquets) pour un trafic donné dans un réseau à commutation de paquets [FV90].

De même, des résultats issus des techniques utilisées dans les couches adaptation d'ATM sont directement applicables à la gestion des ressources mémoires pour les applications multimédias [GK96]. Ainsi, le service CBR dans ATM permet le transfert de données avec un débit crête garanti et une gigue maximale donnée (que nous noterons j). Si l'émetteur transmet à cadence fixe des données (avec un débit d) en utilisant ce service et que l'on souhaite restituer chez le récepteur les données avec cette même cadence, alors il est nécessaire d'absorber la gigue j générée par le réseau. Ce service est offert par les couches d'adaptation d'ATM qui utilisent, pour ce faire, un tampon dont la dimension est de $2 * j * d$ octets. Cette taille garantit l'absence de famine et de débordement. Cette technique est, bien entendue, applicable partout, dans une application multimédias, où une gigue doit être absorbée (exemple : lecture de fichiers depuis un disque). Ce type de résultat est donc important pour le concepteur d'applications ou de systèmes multimédias.

Plus généralement, il existe plusieurs travaux traitant des algorithmes d'ordonnancement de paquets dans les réseaux à commutation où les besoins en mémoire ont été évalués pour chaque politique d'ordonnancement [GP96, PNL96].

Les projets ci-dessus traitent des mécanismes de réservation. Pour ce qui est de l'allocation de la mémoire, celle-ci est presque toujours statique. Une allocation statique est satisfaisante dans une application comme la vidéo-conférence. En effet, les besoins en mémoire de cette application peuvent être déterminés lors de la conception de celle-ci. Pour l'application de télévision interactive, l'allocation statique pose des problèmes de dimensionnement. En effet, il

n'est pas possible de connaître le nombre de flux a priori. L'utilisation d'une allocation statique implique, dans ce cas, une sur-réservation possible de la mémoire ; d'où l'intérêt de disposer ici d'une gestion dynamique de la mémoire.

Or, la gestion dynamique de la mémoire dans les applications temps réel est un sujet étudié depuis longtemps [Bak78, Li90, GN94, Nil94]. Les problèmes habituellement abordés dans ce contexte résident dans la construction d'allocateurs mémoire dont le comportement temporel est prédictible (en général déterministe) ainsi que dans l'ordonnancement conjoint des applications et d'un éventuel ramasse-miettes. Par exemple, Henrikson propose une solution pour ordonnancer le ramasse-miettes et les tâches applicatives grâce à Rate Monotonic [Hen97].

Ces solutions ne sont toutefois pas complètement satisfaisantes pour les applications multimédias. En effet, elles ne prennent pas en compte leurs caractéristiques (besoin de garanties probabilistes, présence de dépendances entre les tâches qui se partagent des tampons, etc). Hormis quelques propositions, telles que celle de Johnstone qui a étudié des gestionnaires de mémoire probabilistes pour les applications temps réel non fortement contraint [Joh97] ou celle de Nilsen proposant une machine virtuelle Java offrant des services temps réel [Nil98], à notre connaissance, il n'existe pas vraiment de travaux sur la gestion dynamique de la mémoire pour les applications multimédias.

Chapitre 7

Conclusions

Dans ce rapport, nous avons présenté les principales solutions qui ont été proposées pour la gestion des ressources dans les systèmes multimédias répartis. Nous avons cité des travaux relatifs à la gestion des ressources processeur, des ressources réseau, des disques et de la mémoire. Ces solutions s'inspirent des algorithmes d'ordonnancement précédemment proposés pour les systèmes temps réel.

Ces domaines ont bénéficié d'un effort important de la part de la communauté scientifique, et il existe aujourd'hui des solutions pour un grand nombre des besoins exprimés, exception faite du support des contraintes statistiques et des flux de données à débit variable.

Nous estimons toutefois qu'ils sont insuffisants pour construire des applications multimédias du fait de l'absence de leur intégration satisfaisante dans un modèle global des ressources.

Chaque mécanisme propose ses propres abstractions. Le développement d'une application oblige alors le concepteur à traduire plusieurs fois la QoS requise par son application en fonction des abstractions associées à chaque ressource. A titre d'exemple, une application qui doit afficher des images de taille fixe à un rythme donné contraint le concepteur à traduire la taille de l'image en débit pour la gestion des ressources réseau, la cadence d'affichage en échéance pour le processeur, etc. L'absence d'un modèle global de haut niveau implique donc un effort important de la part du concepteur. Elle impose que le concepteur dispose de compétences étendues puisqu'il doit connaître pour chaque ressource les abstractions utilisées ainsi que le fonctionnement des directives de gestion de ressources qu'il doit invoquer. Cette difficulté est amplifiée par le fait qu'il n'est pas possible de gérer chaque ressource de façon indépendante. Ainsi, le choix d'un algorithme d'ordonnancement pour les opérations d'entrée/sortie sur un disque induit l'utilisation d'une quantité de mémoire donnée.

Il existe toutefois des propositions qui traitent cette difficulté. Nous citerons pour exemple les nombreux modèles à base de graphes flots de données ou pipe-lines [And93, JB95, MNCK99] qui sont des paradigmes très répandus en multimédias. D'autres proposent l'utilisation d'objets ou de composants pour modéliser les différentes éléments d'un système (ressources, applications, périphériques, etc) [JLDB95, BS98, DLRS99].

Si les mécanismes de gestion de ressources pour les applications multimédias sont relativement matures dans le cadre de leurs utilisations actuelles, nous assistons depuis quelques temps à l'émergence de nouveaux contextes qui reposent le problème de la gestion des ressources dans des conditions parfois bien différentes. Ainsi, de nouvelles contraintes de synchronisation voient le jour avec l'utilisation de plus en plus courante de mondes virtuels multi-acteurs.

De même, de nouvelles contraintes sur le système apparaissent. L'utilisation de systèmes étendus tels qu'Internet (exemple : pour des jeux multimédias répartis) pose le problème du

passage à l'échelle des mécanismes proposés par le passé. Ou encore, avec l'acceptation par le grand public des PAD (*Personal Access Device*), le support des systèmes possédant une quantité limité de ressources mémoire et processeur devient d'actualité. C'est notamment le cas pour l'industrie du téléphone mobile où, avec le futur déploiement de la norme UMTS (*Universal Mobile Telephone Service*) [rGPP99], l'utilisation des applications multimédias devrait se développer.

Dans tous ces nouveaux contextes d'utilisation des applications multimédias, les nouvelles contraintes imposées sur les systèmes et les applications, risquent fort de remettre en cause certains des choix effectués et validés par le passé.

Bibliographie

[Alm97] W. Almesberger. Atm on linux - the 4rd year. in the proceedings of 4th International Linux Kongress, March 1997.

[And93] D. P. Anderson. MetaScheduling for Distributed Continuous Media. *ACM Trans. on Computer Systems*, 11(3):226–252, 1993.

[Bak78] H. G. Baker. List processing in real-time on a serial computer. *Communications of the ACM*, 21(4):280–94, 1978.

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC2475 : An architecture for Differentiated Services. Network Working Group, pages 1-33, December 1998.

[BCM+96] V. Baiceanu, C. Cowan, D. McNamee, C. Pu, and J. Walpole. Multimedia Applications Require Adaptive CPU Scheduling. Workshop on Resource Allocation Problems in Multimedia Systems, Washington DC, December 1996.

[BDS96] I. Busse, B. Deffner, and H. Schulzrinne. Dynamic QoS Control of Multimedia Applications based on RTP. *Computer Communications*, 19(1):49–58, January 1996.

[Bla76] J. Blazewicz. Scheduling Dependant Tasks with Different Arrival Times to Meet Deadlines. In. Gelende. H. Beilner (eds), Modeling and Performance Evaluation of Computer Systems, Amsterdam, Noth-Holland, 1976.

[BMR90] S.K. Baruah, A.K. Mok, and L.E. Rosier. Preemptively scheduling hard real-time sporadic tasks on one processor. pages 182–190. 11th Real time System Symposium, December 1990.

[BNBH98] S. Brandt, G. Nutt, T. Berk, and M. Humphrey. Soft Real-Time Application Execution with Dynamic Quality of Service Assurance. International Workshop on Quality of Service (IWQOS'98), Napa - CA, May 1998.

[Boc95] S. Bocking. Communication Performance Models. Technical report, TR-95-013, International Computer Science Institute, Berkeley, March 1995.

[BPG96] M. M. Buddhikot, G. M. Parulkar, and R. Gopalakrishnan. Scalable Multimedia-On-Demand via World-Wide-Web (WWW) with QoS Guarantees. pages 23–26. Sixth International Workshop on Network and Operating System Support for Digital Audio and Video, NOSSDAV'96, Zushi, Japon, April 1996.

[BR96] D. W. Brubeck and L. A. Rowe. Hierarchical Storage Management in a Distributed VOD System. *IEEE Multimedia*, 3(3):37–47, Fall 1996.

[Bra97a] T. Braun. Internet Protocols for Multimedia Communications. Part I. IPng - The Foundation of Internet Protocols. *IEEE Multimedia, July-September*, 4(3):85–90, 1997.

[Bra97b] T. Braun. Internet Protocols for Multimedia Communications. Part II. Resource Reservation, Transport and Applications Protocols. *IEEE Multimedia, October-December*, 4(4):74–82, 1997.

[BS98] G. Blair and J. B. Stefani. *Open Distributed Processing and Multimedia.* Addison-Wesley, 1998.

[BTW94] J. C. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the Internet. pages 58–67. In Proc. ACM SIGCOMM'94, London, UK, September 1994.

[BZ93] M.C. Buchanan and P.T. Zellweger. Automatically Generating Consistent Schedule For Multimedia Applications. *Multimedia Systems Journal*, 1(2):55–67, 1993.

[CAH98] A. Campbell, C. Aurrecoechea, and L. Hauw. A Survey of QoS Architectures. *Multimedia Systems Journal, Special Issue on QoS Architecture*, 6(3):138–151, May 1998.

[Can94] S. Mc Canne. Vic and RTPv2. Technical report, Audio-Video Transport Working Group, IETF 31, San Jose, CA, December 1994.

[CCH98] F. Cottet, M. Courtes, and M. Holle. Traitement de la gigue temporelle pour les ordonnancements temps réel par échéance. pages 63–77. Real Time Systems, Paris, janvier 1998.

[CCI90] CCITT. Recommendations H.261 : Video codec for audiovisual services at p*64 kb/s. White book, 1990.

[CM94] C. Cardeira and Z. Mammeri. Ordonnancement de tâches dans les systèmes temps réel et répartis. *APII*, 28(4):353–384, 1994.

[CM97] G. Coulson and A. Mauthe. Scheduling and Admission Testing for Jitter Constrained Periodic Threads . *ACM Multimedia Systems Journal*, 5(5):337–346, 1997.

[CPS+95] S. Cen, C. Pu, R. Staehli, C. Cowan, and J. Walpole. A Distributed Real-Time MPEG Video Audio Player . Appeared in the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95). Durham, New Hampshire, USA, April 1995.

[CSB90] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic Scheduling of Real-time Tasks Under Precedence Constraints . *Real Time Systems, The International Journal of Time-Critical Computing Systems*, 2(3):181–194, September 1990.

[CT90] D. D. Clark and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. pages 200–208, Philadelphia, PA, September 1990. ACM SIGCOMM Symposium on Communications Architectures and Protocols.

[Dio95] C. Diot. Adaptive Applications and QoS Guaranties. Invited paper in the MmNet'95 International Conference on Multimedia Networking. Aizu-Wakamatsu, Japan, September 1995.

[DLRS99] I. Demeure, L. Leboucher, N. Rivierre, and F. Singhoff. Modélisation et support d'applications multimédias réparties. *Calculateurs parallèles, Réseaux et systèmes répartis.*, 11(2):161–191, septembre 1999.

[For96] ATM Forum. Traffic Management Specification, Version 4.0. Document References : af-tm-0056.000, April 1996.

[FSVW96] M. Fry, A. Seneviratne, A. Vogeland, and V. Witina. Delivering QoS controlled Continuous Media on the World Wide Web. pages 45–53. in Proceedings of the 4th international IFIP Workshop on Quality of Service, Paris, March 1996.

[FV90] D. Ferrari and D. C. Verma. Buffer Space Allocation for Real-Time Channels in a Packet Switching Network. Technical report, numéro TR-90-022, International Computer Science Institute, Berkeley, June 1990.

[Gaf99] J. Gafsi. *Design and Performance of Large Scale Video Servers.* Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications de Paris, novembre 1999.

[Gal95] B. O. Gallmeister. *POSIX 4 : Programming for the Real World* . O'Reilly and Associates, January 1995.

[GB97] D. J. Gemmell and C. G. Bell. Noncollaborative Telepresentations Come of Age. *Communications of the ACM*, 40(4):79–89, April 1997.

[GC92] J. Gemmell and S. Christodoulakis. Principles of delay sensitive multimedia data storage and retrieval. *ACM Transactions on Information Systems*, 10(1):51–90, January 1992.

[GC95] B. Goodheart and J. Cox. *The Magic Garden Explained: The Internals of UNIX SYSTEM V Release 4.* Prentice Hall, 1995.

[GK96] M. Gagnaire and D. Kofman. *Réseaux Haut Débit : réseaux ATM, réseaux locaux, réseaux tout-optiques.* Masson-Inter Editions, Collection IIA, 1996.

[GN94] H. Gao and K. Nilsen. Reliable general purpose dynamic memory management for real-time systems. Technical Report TR94–09, Iowa State University, July 1994.

[GP96] R. Gopalakrishnan and G. M. Parulkar. Bringing Real-time Scheduling Theory and Practice Closer for Multimedia Computing. Proceedings of the ACM SIGMETRICS, Conference on Measurement and Modeling of Computer Systems, Philadelphia, May 1996.

[GRS96] L. George, N. Rivierre, and M. Spuri. Preemptive and Non-Preemptive Real-time Uni-processor Scheduling. INRIA Technical report number 2966, 1996.

[GVK+95] D. J. Gemmell, H. M. Vin, D. D. Kamdlur, P. V. Rangan, and L. A. Rowe. Multimedia Storage servers : a tutorial. *IEEE Computer magazine*, 28(5):40–49, May 1995.

[HBD98] A. Hafid, G. Bochmann, and R. Dssouli. Distributed Multimedia Application and Quality of Service : a Review. *Electronic Journal on Networks and Distributed Processing*, 2(6):1–50, 1998.

[Hen97] R. Henriksson. Predictable Automatic Memory Management for Embedded Systems. OOPSLA '97 Workshop on Garbage Collection and Memory Management, October 1997.

[HLH96] C. C. Han, K. J. Lin, and C. J. Hou. Distance-constrained Scheduling and Its Applications to Real-time Systems. *IEEE Transactions on computers*, 45(7):814–826, July 1996.

[HLR97] J.F. Hermant, L. Leboucher, and N. Rivierre. Real time Fixed and Dynamic Priority Driven Scheduling Algorithms : Theory and Experience. Rapport de recherche de l'INRIA numéro 3081, 1997.

[ISO94] ISO/IEC. JTC 1/SC 29/WG 11 number 702 Rev. Information Technology - Generic Coding of Moving Pictures and associated Audio, recommandation H 262.". Draft International Standard, Paris 25 March, 1994.

[JB95] K. Jeffay and D. Bennett. A Rate-Based Execution Abstraction For Multimedia Computing. *In Lectures Notes in Computing Science, T. D. C. Little and R. Gusella, Springer-Verlag, Heidelberg*, 1018:64–75, April 1995.

[JG99] K. Jeffay and S. Goddard. A Rate-Based Execution Model. The 20th IEEE Real-Time Systems Symposium, Phoenix, Arizona USA, December 1999.

[JLDB95] M.B. Jones, P.J. Leach, R.P. Draves, and J.S. Barrera. Modular Real-time Resource Management in the Rialto Operating System. Fifth Workshop on Hot Topics in Operating Systems (HoT-OS-V), May 1995.

[Joh97] M. S. Johnstone. *Non-Compacting Memory Allocation and Real-Time Garbage Collection*. PhD Thesis, University of Texas at Austin, December 1997.

[JRR97] M. Jones, D. Rosu, and M. Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. 16th ACM Symposium on Operating Systems Principles in Saint-Malo (SOSP'97) - France, October 1997.

[JSS92] K. Jeffay, D. L. Stone, and F. D. Smith. Kernel Support for Live Digital Audio and Video. *Computer Communications*, 15(6):388–395, August 1992.

[KBB00] H. Kosch, K. Breidler, and L. Bszrményi. A Comparative Study of Selected Parallel Video Servers. pages 669–673. In DEXA 2000 Workshop Proceedings, IEEE CS Press, W10, Greenwich (UK), September 2000.

[LA98] M. Laubach and J. Alpern. RFC2225 : Classical IP and ARP over ATM. Network Working Group, pages 1-17, April 1998.

[Leb98] L. Leboucher. *Algorithmique et Modélisation pour la Qualité de Service des Systèmes Répartis Temps Réel.* Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications de Paris, septembre 1998.

[Leh90] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. pages 201–209. in Proc. 11th IEEE Real Time Systems Symposium, Lake Buena Vista, December 1990.

[Li90] K. Li. Real-Time Concurrent Collection in User Mode. OOPSLA/ECOOP '90 Workshop on Garbage Collection in Object-Oriented Systems, October 1990.

[LL73] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[LM80] J. Y. T. Leug and M. L. Merril. A note on preemptive scheduling of periodic real time tasks. *Information Processing Letters*, 11(3):115–118, 1980.

[MNCK99] S. Mitchell, H. Naguib, G. Coulouris, and T. Kindberg. A qos support framework for dynamically reconfigurable multimedia applications. pages 17–30. in Proc. of the Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems, Helsinki, Finland, June 1999.

[Mok83] A.K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real Time Environnement.* PhD Thesis, Departement of electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, May 1983.

[MST94] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Applications. In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, May 1994.

[Nil94] K. D. Nilsen. Reliable real-time garbage collection of C++. *Computing Systems*, 7(4), 1994.

[Nil98] K. Nilsen. Adding Real-Time Capabilities to Java. *Communications of the ACM*, 6(41):49–56, June 1998.

[NL97] J. Nieh and M. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. 16th ACM Symposium on Operating Systems Principles in Saint-Malo (SOSP'97) - France, October 1997.

[NRW94] A.L. Narasimha, Reddy, and J. Wyllie. I/O Issues in a Multimedia System. *IEEE Computer*, 27(3):69–74, March 1994.

[PGK88] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). pages 109–116. June 1988.

[PNL96] I. R. Philp, K. Nahrstedt, and J. W. S. Liu. Scheduling and buffer management for soft real-time VBR traffic in packet switched networks. pages 143–152, Minneapolis, Minnesota, October 1996. Proceedings, 21st Conference on Local Computer Networks.

[RCC+94] P. Robin, G. Coulson, A. Campbell, G. Blair, and M. Papathomas. Implementing a qos controlled atm based communications system in chorus. Proceedings of the 4th International Workshop on Protocols for High Performance Networks, Vancouver, Canada, 1994.

[rGPP99] 3rd Generation Partnership Project. Technical Specification Group Services and Systems Aspects : General UMTS Architecture. Version 3.0.1. Document References : 3G TS 23.101, June 1999.

[Ric98] P. Richard. Télévision numérique : comment choisir son "bouquet". *Science et Vie, 964:117-123*, janvier 1998.

[Riv98] N. Rivierre. *Ordonnancement temps réel centralisé, les cas préemptifs et non préemptifs*. Thèse de doctorat, Université de Versailles Saint Quentin, février 1998.

[RV93] P. V. Rangan and H. M. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE trans. on Knowledge and Data Engineering*, August 1993.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC1889 : RTP : A Transport Protocol for Real-Time Applications. Network Working Group, pages 1-75, January 1996.

[SGP95] M. Saksena, R. Gerber, and W. Pugh. Parametric Dispatching of Hard Real-time Tasks. *IEEE Trans. on Computers*, 44(3):471–479, 1995.

[SJ87] K. Sevcik and M. Johnson. Cycle Time Properties of the FDDI Token Ring Protocol. *IEEE Transactions on Software Engineering*, 13(3):376–385, March 1987.

[SM96] P. Sijben and S. J. Mullender. Quality of Service in Distributed Multimedia Systems. in Trends in distributed systems, Springer Lectures Notes on computing systems, TREVS 1161, 1996.

[SN95] R. Steinmetz and K. Nahrstedt. *Multimedia : Computing, communicating and applications*. Prentice Hall, innovative technology series, 1995.

[SP94] A. Shah and G. Pamakrishnan. *FDDI, réseaux haut débit*. Collection systèmes distribués, MASSON, 1994.

[SPG97] S. Shenker, C. Partridge, and R. Guerin. RFC2212 : Specification of Guaranted Quality of Service. Network Working Group, pages 1-19, September 1997.

[SRL90] L. Sha, R. Rajkumar, and J.P. Lehoczky. Priority Inheritance Protocols : An Approach to real-time Synchronization. *IEEE Transactions on computers*, 39(9):1175–1185, 1990.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. RFC2326: Real Time Streaming Protocol (RTSP), April 1998.

[SSL89] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic Task Scheduling for Hard-real-time Systems. *The Journal of Real Time Systems*, 1:27–60, 1989.

[SSNB95] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo. Implications of Classical Scheduling Results For Real-Time Systems. *IEEE Computer*, 28(6):16–25, June 1995.

[Sus00] J.F. Susbielle. *Internet, multimédia et temps réel*. Editions Eyrolles, février 2000.

[TM89] H. Tokuda and C. W. Mercer. ARTS kernel: A distributed real-time kernel. *ACM Operating Systems Review*, 23(3), 1989.

[TTCM92] H. Tokuda, Y. Tobe, S. Chou, and J. Moura. Continuous Media Communication with Dynamic QoS Control Using ARTS with an FDDI Network. pages 88–98. ACM SIGCOMM'92, August 1992.

[Vet95] R. J. Vetter. ATM concepts, architectures, and protocols. *Communications of the ACM*, 38(2):30–38, February 1995.

[VKBG95] A. Vogel, B. Kerhervé, G. Von Bochmann, and J. Gecsei. Distributed Multimedia and QoS : A Survey. *IEEE Multimedia*, 2(2):10–19, Summer 1995.

[VT97] L. Vega and J.P. Thomesse. Vers une caracterisation temporelle des profils de communication. pages 81–97. Real Time Systems, Paris , janvier 1997.

[WG94] S. Wray and T. Glauert. Networked Multimedia : The Medusa Environment. *IEEE Multimedia*, 1(4):54–63, 1994.

[Wic97] I. J. Wickelgren. The facts about FireWire. *IEEE Spectrum*, 34(4):19–25, April 1997.

[Wro97] J. Wroclawski. RFC2210 : The Use of RSVP with IETF Integrated Services. Network Working Group, pages 1-31, September 1997.

[Zha93] H. Zhang. *Service Disciplines For Packet-Switching Integrating-Services Networks*. PhD Thesis, University of California at Berkeley, November 1993.

[ZS95] Q. Zheng and K. G. Shin. Synchronous Bandwidth Allocation in FDDI Networks. *IEEE Transaction on Parallel and Distributed Systems*, 6(12):1332–1338, December 1995.