Ordonnanceurs hiérarchiques adaptables

Charles Clément, Bertil Folliot

Ordonnanceurs hiérarchiques adaptables

Charles CLÉMENT, Bertil FOLLIOT
Laboratoire d'informatique de Paris 6
charles.clement@lip6.fr, bertil.folliot@lip6.fr

Résumé : Les systèmes d'exploitation à temps partagé offrent à l'utilisateur un confort d'utilisation en permettant de simuler l'exécution simultanée de plusieurs programmes. Pour cela le système traite équitablement et à tour de rôle chacune des tâches pendant un quantum de temps défini. L'inconvénient est que l'utilisateur peut exécuter différentes catégories de programmes ne reposant pas sur les mêmes critères. Les applications multimédias, comme la lecture d'un fichier vidéo, peuvent souffrir de cette équité dès lors qu'un certain nombre de programmes s'exécutent. Lorsqu'elles ne possèdent pas les ressources suffisantes elles deviennent inutilisables : les images ne sont plus synchronisées avec la bande son et l'affichage devient saccadé. Cependant, toutes les applications ne nécessitent pas de contraintes temporelles et, par exemple, les traitements de données ne sont pas altérés si les résultats sont produits avec du retard. Nous présentons nos travaux sur la plate forme MVV-Bossa qui fournit à l'utilisateur un moyen de casser l'équité entre les différentes tâches et donc lui permet de mettre l'accent sur l'exécution d'un programme particulier pour obtenir la qualité de service qu'il désire.

Mots-clés : Système d'exploitation, ordonnancement, qualité de service, multimédia.

1 INTRODUCTION

Les systèmes d'exploitation fournissent les abstractions nécessaires aux applications pour exploiter le matériel et les périphériques de l'ordinateur. Dans un système multitâche, l'utilisateur a la possibilité de lancer plusieurs applications à la fois et a l'impression qu'elles s'exécutent en même temps.

Les systèmes d'exploitation utilisés dans les ordinateurs de bureau sont dits multitâches. Cela est justement lié à la politique d'allocation du processeur : l'ordonnancement. Au lieu d'exécuter une tâche jusqu'à ce que celle-ci se termine, le système va partager les temps d'utilisation du processeur entre les différentes tâches en utilisant des quanta de temps suffisamment courts pour donner l'impression que toutes les tâches s'exécutent simultanéments. Cette politique d'ordonnancement reposant sur l'utilisation de tranches de temps d'exécution est caractéristique des systèmes dits à temps-partagé (**time-sharing**). Ces derniers ont pour but de fournir une répartition équitable du temps de calcul entre toutes les applications, pour garantir leur exécution.

Le fait de partager les temps d'exécution permet donc d'obtenir un confort d'utilisation. Seulement, cette répartition équitable ne tient pas compte des différentes classes d'application. Il existe ainsi plusieurs classes définies par le type de traitement qu'elles effectuent, le résultat produit et les contraintes d'exécution. La première catégorie est appelée **batch processing** et correspond au traitement de tâches non interactives. Elle regroupe les tâches de type calcul scientifique ou de compilation qui effectuent des calculs sur un ensemble de données sans intervention extérieure et fournissent un résultat à la fin du traitement. Ce genre d'application est très gourmand en ressources matérielles car les données étant déjà prêtes, elles calculent intensivement. La seconde catégorie, par opposition, décrit les applications interactives (**interactive processing**), c'est à dire les applications dont l'exécution est dépendante de l'interaction avec l'utilisateur. On y retrouve les éditeurs de texte, les navigateurs internet ou tout autre logiciel qui s'exécute en fonction des événements produits par l'utilisateur. Ces applications ne nécessitent pas de ressources de calcul importante, mais des temps de réponse assez courts.

Le système ne fait pas de différence entre ces classes de processus [Etsion, 2004] car il n'a aucun moyen de les dissocier et, en conséquence, il répartit équitablement la charge de travail. Un problème se pose lorsqu'une ou plusieurs tâches consomment beaucoup de ressource de calcul. Si l'utilisateur est en train d'utiliser un éditeur de texte, cela se traduira par des temps de réponse un peu plus long. Mais si l'utilisateur utilise une application multimédia, comme par exemple un lecteur de fichier vidéo, le partage équitable du processeur se traduit par un affichage saccadé, car l'application ne dispose pas des ressources nécessaires à son bon déroulement. L'utilisateur ne dispose alors d'aucun moyen lui permettant d'influer sur l'allocation des ressources, si ce n'est d'arrêter les processus en tâche de fond, perdre le travail déjà effectué et les ré-exécuter ultérieurement.

Nous résolvons ce problème en réalisant un système dans lequel une application ou un utilisateur peut adapter l'ordonnanceur à ses besoins. Ainsi une application ayant des contraintes temporelles peut, en fonction de la qualité offerte à l'utilisateur, augmenter (ou diminuer) son taux d'allocation de temps processeur. Le partage équitable des temps de calcul entre toutes les tâches est ainsi annulé, pour pouvoir offrir à l'utilisateur une qualité de service appropriée à l'application. L'utilisateur peut ainsi exécuter des applications nécessitant des temps de latence courts, ou des contraintes temporelles alors que des calculs intensifs s'exécutent en arrière-plan.

Le plan de cet article est le suivant. La deuxième section présente d'autres projets permettant d'intégrer de la qualité de service pour les applications multimédias dans les systèmes d'exploitation. La troisième section décrit l'outil Bossa [Barreto, 2001] qui sert de support à nos travaux. La quatrième section présente l'environnement actif, la MVlet Bossa, la cinquième section décrit un exemple d'application que nous avons réalisé et enfin la sixième une évaluation du système.

2 ÉTAT DE L'ART

Dans cette section sont présentés les systèmes d'exploitation mettant en œuvre des mécanismes de réservation de ressources, puis un ordonnanceur prenant en compte les caractéristiques des tâches multimédias.

2.1 Système basé sur la réservation de ressource

Les systèmes NEMESIS et QLinux sont tous deux des systèmes dont le but est de pouvoir fournir aux applications de la qualité de service.

NEMESIS [Reed, 1997, Leslie, 1996] est un système d'exploitation développé dans le but de fournir aux applications des garanties sur l'utilisation des ressources. Cela consiste à mettre en œuvre un système capable d'assurer une qualité de service aux applications nécessitant des ressources dans un modèle semi-périodique, en l'assimilant à une tâche ayant des contraintes temporelles strictes. Pour cela, NEMESIS permet d'attribuer des durées d'utilisation précises sur l'allocation des ressources système comme le processeur, la bande passante au niveau du réseau et des disques. Le principal inconvénient de cette approche est de devoir spécifier précisément les contraintes temporelles des applications. Dans le cas précis des applications multimédias, par exemple la lecture d'une vidéo, il est très difficile de spécifier ces paramètres, car ils sont dépendant de la machine, du contexte, des propriétés du fichier, telle que l'algorithme de compression utilisée, la résolution d'affichage… Ces caractéristiques ne peuvent être connues à l'avance et entraînent une étude très fastidieuse, à appliquer au cas par cas.

QLinux [Sundaram, 2000] est un système d'exploitation orienté multimédia basé sur le noyau Linux dans lequel il est possible, à la manière de NEMESIS, d'offrir aux applications de la qualité de service.

Ce noyau prend en compte le fait que les systèmes actuels exécutent avec la même politique différentes classes d'applications. Les trois classes d'applications sont celle ayant des contraintes strictes, pour lesquelles il faut garantir des délais d'exécution ; celles ayant des contraintes souples nécessitant de la qualité de service tel un délai maximum ou un débit constant ; et celles ne nécessitant pas de garanties.

L'algorithme repose sur une hiérarchie dans laquelle les applications ou classes d'application possèdent un taux d'utilisation du processeur. Cette hiérarchie est représentée sous la forme d'un arbre, dans laquelle les classes d'applications sont les feuilles. Tout nœud qui n'est pas une feuille est une agrégation de classes d'applications qui possède un poids, représentant le pourcentage d'utilisation processeur qui lui est alloué.

Cette technique permettant d'allouer de manière prédictible le processeur ne nécessite plus de spécifier les contraintes temporelles de chaque application. En revanche, il faut mettre en place un gestionnaire, soit directement au niveau de l'application, soit au niveau du système, capable de créer la hiérarchie et d'affecter les poids. Ce qui pose un problème plus général car le système ignore ce que fait l'application et ne peut savoir à l'avance la quantité de ressources dont elle aura besoin.

2.2 SMART

SMART [Nieh, 2003] (A scheduler for multimedia and real-time applications) est un ordonnanceur dont le but est de prendre en compte les spécificités des applications multimédias en terme d'utilisation de ressources, de pouvoir adapter l'application en fonction de la charge du système et d'intégrer les préférences de l'utilisateur. Deux interfaces permettant d'ajuster la politique d'ordonnancement ont été définies pour prendre en compte les directives du développeur d'applications et les choix de l'utilisateur.

Les applications ne possèdent pas d'informations sur la charge du système, et en conséquence, il est très difficile de savoir combien de temps prendra l'exécution d'un calcul. Il est possible de spécifier dans le code source une contrainte temporelle correspondant à la durée d'exécution d'une partie du code de l'application et un code alternatif associé. L'ordonnanceur peut alors notifier l'application lorsque la demande ne peut pas être satisfaite au vu de la charge du système et donc exécuter la section de code alternative.

La seconde interface est destinée à l'utilisateur. En fonction de l'importance qu'il attache aux applications et de ses préférences, SMART permet de favoriser l'attribution des ressources pour améliorer les performances d'une application. Pour cela, deux paramètres sont utilisés, *priorité* et *taux de partage* mais ne sont pas obligatoires. La priorité est utilisée lorsque deux processus sont en concurrences pour l'attribution de ressources. Celui qui possède la plus haute priorité se verra attribuer la ressource. Lorsque deux processus possèdent la même priorité, alors la répartition se fera proportionnellement au paramètre *taux de partage*.

2.3 Synthèse

Les différentes solutions étudiées proposent des interfaces permettant à l'application d'effectuer de la réservation de ressource. Pour ce qui est de l'utilisation processeur, on remarque que les modèles de processus tendent à se rapprocher des modèles périodiques, et essayent d'associer une période et une échéance aux processus. Seulement, pour les applications traditionnelles, et dans le cas des applications multimédias, il est très difficile de représenter leurs comportements dans un modèle périodique, qui va dépendre des données à traiter, de la position dans le flux (typiquement dans le cas d'un codage/décodage utilisant un taux variable comme les fichiers mp3), de la plate-forme, de la synchronisation de flux…

C'est pourquoi notre approche, ordonnanceur hiérarchique adaptable se base sur la mise en place de niveau de priorité au sein d'une hiérarchie [Regehr, 2001], ce qui permet d'éviter l'étude statique du comportement des applications à traiter.

3 LE FRAMEWORK BOSSA

Bossa [Barreto, 2001] est un outil développé à l'École des Mines de Nantes, dont le but est de faciliter l'implémentation et l'intégration de nouvelles politiques d'ordonnancement. Le développement d'un nouvel ordonnanceur est une tâche compliquée car cela nécessite de comprendre et de modifier de multiples parties du code d'un noyau. Comme la politique d'ordonnancement des tâches est mise en œuvre dans de multiples routines, les systèmes ne définissent pas d'interfaces permettant de modifier celle-ci aisément. Bossa est une architecture logicielle basé sur un Domain Specific Language (DSL) qui permet d'utiliser des abstractions de haut niveau, un compilateur et des outils pour mettre en place ces politiques.

L'ordonnanceur étant une fonctionalité interne du système d'exploitation, il est nécessaire d'utiliser le compte du super-utilisateur (root) pour pouvoir modifier la politique d'ordonnancement.

3.1 Le langage Bossa

Bossa possède un langage dédié [Barreto, 2002] qui permet de développer un ordonnanceur simplement. Ce langage fournit au programmeur des déclarations et abstractions spécifiques permettant de manipuler les processus. Une politique d'ordonnancement est composée de trois parties : les déclarations, les routines associées aux événements du noyau et l'interface de l'ordonnanceur.

3.2 Les ordonnanceurs

Bossa propose par défaut différents ordonnanceurs, parmi lesquels la politique du noyau Linux 2.4, et la politique EDF (Earliest Deadline First), un ordonnanceur temps réel. Une hiérarchie d'ordonnanceur a la structure d'un arbre composé de plusieurs types d'ordonnanceur. Tout nœud qui n'est pas une feuille est appelé un ordonnanceur virtuel, à la différence des feuilles qui sont des ordonnanceurs de processus.

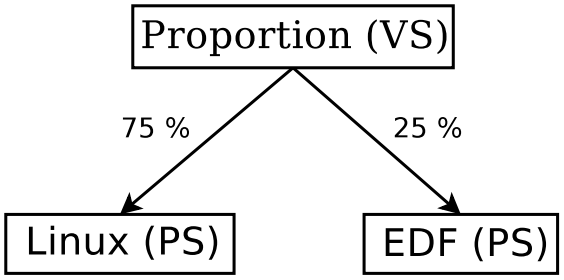

FIG. 1 – Arbre d'ordonnancement.

La figure 1 présente un exemple de hiérarchie dans laquelle le nœud racine est un ordonnanceur virtuel qui répartit la charge selon les paramètres de chaque ordonnanceur de processus, avec un taux de 25% pour EDF et de 75% d'utilisation processeur pour Linux.

3.2.1 Les ordonnanceurs virtuels

Comme expliqué plus haut, ceux-ci servent à ordonnancer non pas des processus, mais des ordonnanceurs. Dans une hiérarchie, chaque nœud non terminal est un ordonnanceur virtuel.

Proportion Cet ordonnanceur repose sur l'utilisation de taux, c'est un ordonnanceur non préemptif. Chaque fils possède un pourcentage qui détermine le taux processeur minimum qui lui est attribué. Un test est effectué lors de l'attachement d'un ordonnanceur pour vérifier que la somme des taux ne dépasse pas 100%.

Fixed Priority Comme son nom l'indique, cet ordonnancement est effectué à l'aide de priorités statiques. Cet ordonnanceur est préemptif, c'est à dire qu'un processus s'exécutant peut-être évincé si il existe un processus prêt de plus haute priorité. Il est à noter que l'utilisation de cet ordonnanceur peut conduire à des cas de famines, lorsqu'un processus ne peut s'exécuter car d'autres, de plus haute priorité sollicitent le processeur sans relâche.

3.2.2 Ordonnanceur de processus

Il existe plusieurs ordonnanceurs de processus fournis avec Bossa tels BVT (Borrowed Virtual Time) et BEST (Best-Effort Scheduler enhanced for soft real-time timesharing) ainsi que des ordonnanceurs temps réel. On s'intéressera plus particulièrement à celui fourni avec le noyau Linux 2.4, qui utilise l'algorithme du tourniquet (Round Robin). La liste des processus prêt est circulaire et chaque processus exécuté pendant un quantum de temps est ensuite placé à la fin de la file.

4 L'ENVIRONNEMENT ACTIF : VMLET BOSSA

La Machine Virtuelle Virtuelle permet de faire le lien entre le noyau du système d'exploitation et l'application. Grâce à l'environnement actif (la MVlet Bossa), il est

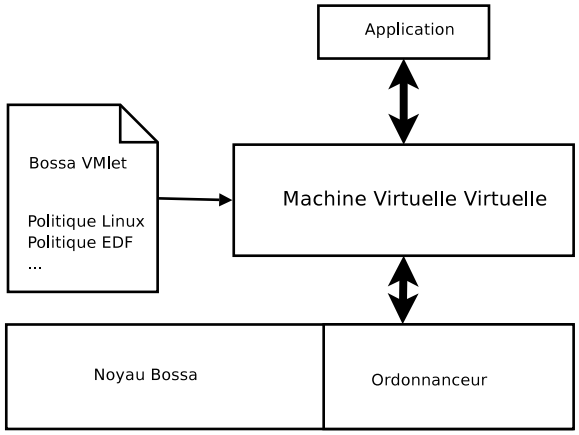

FIG. 2 – Architecture du système MVV-Bossa.

possible de modifier le système pendant l'exécution de l'application.

4.1 La Machine Virtuelle Virtuelle

La Machine Virtuelle Virtuelle [Folliot, 1998] est une plate forme d'exécution dans laquelle il est possible de spécialiser et d'adapter l'environnement pour les besoins de l'application. Pour cela, elle ne fournit que le strict nécessaire en termes d'abstraction et un mécanisme permettant de mettre à jour la plate forme grâce au chargement de descriptions de machines virtuelles.

4.2 La MVlet

Une MVlet est une description de Machine Virtuelle. Elle permet de modifier dynamiquement la plate forme d'exécution. L'utilisation seule de Bossa permet de modifier la politique d'ordonnancement mais elle est alors définie statiquement lors de la phase de compilation. Il est ainsi nécessaire de modifier le code source de l'application et donc de le posséder, ce qui n'est pas toujours le cas lors de l'utilisation de programmes commerciaux, fournis sous la forme de fichiers binaires. En revanche, grâce à la Machine Virtuelle Virtuelle [Piumarta, 2000] il est possible de modifier l'environnement d'exécution et donc le système sans toucher à l'application et de façon transparente pour celle-ci.
Le chargement d'une MVlet (voir figure 2) étend les fonctionnalités du système et dans notre cas offre les abstractions nécessaires au domaine de l'ordonnancement.
La dynamicité introduite offre la possibilité de prendre en compte des paramètres inconnus lors de l'écriture du programme et de réagir en conséquence. On peut donc adapter le système en fonction des conditions d'utilisation.

4.3 Chargement des politiques

Les différentes politiques d'ordonnancement sont compilés sous la forme de modules noyaux, ce sont donc des extensions du système d'exploitation. Pour pouvoir s'en servir, il faut les charger au préalable, sachant que chaque politique est contenue dans un module. Ainsi, pour charger une hiérarchie contenant trois ordonnanceurs, il faut charger un module contenant un ordonnanceur virtuel, soit Fixed_priority ou Proportion (voir section 3.2.1), et un ordonnanceur de processus en plus de l'ordonnanceur par défaut du noyau Linux. Cet ordonnanceur par défaut est indispensable car il gère toutes les applications et processus nécessaires au système.
De manière analogue il faut être en mesure de les décharger lorsqu'ils ne sont plus utilisés. Pour cela il faut d'abord s'assurer que ces ordonnanceurs ne comportent plus de processus actifs. De plus, si à l'issue du déchargement il ne reste plus qu'un seul ordonnanceur de processus, il faut aussi enlever l'ordonnanceur virtuel pour mettre à la racine celui par défaut.

4.4 Interfaces d'accès

Une librairie partagée encapsulant les fonctions qui manipulent l'ordonnanceur est nécessaire pour effectuer les communications avec le système.

4.4.1 Chargement des ordonnanceurs

La première fonction sert à ajouter un ordonnanceur dans la hiérarchie. Lors du premier appel est mis en place un ordonnanceur virtuel qui sera la racine de la hiérarchie d'ordonnancement, auquel est automatiquement lié un premier fils qui est l'ordonnanceur par défaut de Linux.
Ensuite, à chaque ajout d'un nouvel ordonnanceur, il faut spécifier quel est son père dans la hiérarchie et les attributs qui lui sont associés dans l'ordonnanceur père.

4.4.2 Migration des processus

Une fois qu'une hiérarchie est chargée dans le système, les processus sont par défaut ordonnancés par l'ordonnanceur Linux, comme lors d'une utilisation normale. Ainsi, les applications lancées après avoir mis en place une hiérarchie seront toujours par défaut gérées par l'ordonnanceur principal de Linux.
Pour pouvoir migrer un processus il faut tout d'abord récupérer son identifiant au sein du système, le pid (pour identificateur de processus) et connaître l'ordonnanceur cible dans lequel on veut attacher ce processus. Ensuite il faut appeler la fonction d'attachement du processus qui va déconnecter le processus de l'ordonnanceur dans lequel il se trouvait pour l'attacher au nouveau. Ces fonctions sont propres à chaque ordonnanceur car c'est dans ces routines que sont associés et initialisés les attributs des processus, et, comme les attributs sont spécifiques à chaque politique, il n'y a pas de méthode générique.

4.5 Modèle d'application

4.5.1 D´etection du mode d´egrad´e

Lorsque le système dispose d'assez de ressources pour satisfaire toutes les demandes des applications, l'exécution se produit de manière traditionnelle, en utilisant la politique à temps partagée de Linux. Si en revanche l'application visée ne dispose pas des performances nécessaires en raison du nombre d'applications s'exécutant en même temps, elle va donc construire une hiérarchie d'ordonnanceur.

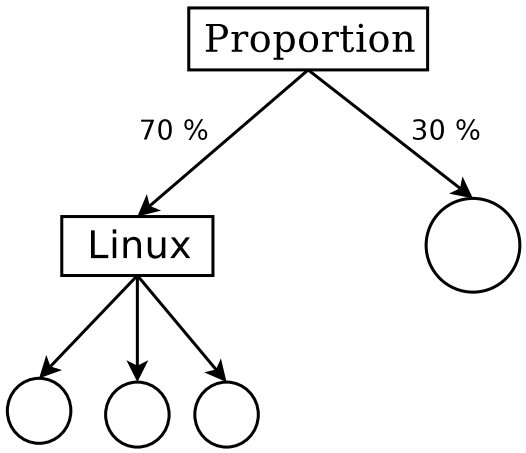

FIG. 3 – Hiérarchie de processus.

Il faut donc être capable de détecter que l'exécution du programme n'est pas satisfaisante. Pour cela l'évolution d'un critère significatif des performances de l'application est étudié au cours du temps. Ce critère est intimement lié à l'application et ne peut donc être défini dans l'absolu, mais au cas par cas.
Une première solution consiste donc à laisser le choix au développeur de définir la valeur seuil au delà de laquelle l'application s'exécute en mode dégradé. Une seconde approche consiste à laisser délibérément le choix à l'utilisateur, en utilisant une interface graphique lui permettant de spécifier l'application qui nécessite de la qualité de service.

4.5.2 Chargement de la hi´erarchie

Une fois détectée l'insuffisance des ressources allouées, l'ordonnanceur virtuel Proportion (voir section 3.2.1) est chargé et il est donc possible d'affecter des taux d'utilisation processeur pour chaque ordonnanceur. Dans le cas d'applications utilisées communément, il n'est pas nécessaire d'utiliser un ordonnanceur de processus à modèle périodique, on peut utiliser un autre ordonnanceur utilisant une politique à temps partagé. Le processus concerné est ensuite migré dans le nouvel ordonnanceur. L'affectation des taux se faisant au sommet de la hiérarchie, il en résulte une structure contenant deux ordonnanceurs utilisant une politique à temps partagé. Le premier contient l'ensemble des tâches du système, alors que le second ne contient que le processus nécessitant la qualité de service.

4.5.3 Affectation des taux

Le fait de dissocier un programme du reste du système implique que le partage équitable entre l'ensemble des tâches n'est plus respecté. Cela étant, en ayant un seul processus dans un ordonnanceur à temps partagé, le taux affecté à l'ordonnanceur ne profitera donc qu'à ce processus. En pratique, par exemple dans le cadre du décodage d'un flux vidéo, un taux de l'ordre de 30% suffit à l'application pour qu'elle ne saccade plus. Cette valeur est bien sûr liée à notre machine de test, et est dépendante de la puissance de calcul de la machine. La figure 1 illustre le partage du processeur, le fils gauche représente l'ordonnanceur linux normal dans lequel les tâches se partagent 70% du processeur alors que le fils droit représente le processus possédant un taux de 30%.

5 APPLICATION À UN FLUX VIDÉO

Une application concrète du modèle exposé est présenté dans cette section. Nous prenons comme exemple un logiciel de lecture vidéo, dont les besoins en terme de ressources de calcul sont constants, car les calculs s'effectuent au fur et à mesure de la lecture.
Le but est de proposer une application réalisant le décodage et l'affichage d'un flux vidéo capable d'adapter le système sous-jacent lorsque ses ressources de calcul sont insuffisantes pour offrir une lecture fluide. Il faut premièrement être capable de quantifier l'exécution, c'est-à-dire trouver un critère qui caractérise la qualité offerte à l'utilisateur.

5.1 Critère associé à la lecture d'une vidéo

Lorsqu'un utilisateur lit un flux vidéo sur un ordinateur, le résultat attendu est une fenêtre qui affiche l'ensemble des données contenues dans le fichier en tenant compte des informations temporelles. Ainsi, un flux vidéo contient comme information la durée de chaque de image. Pour une vidéo dont le nombre d'images par seconde est de 25, chaque image possède une fenêtre temporelle de $1/25 = 0,04$ secondes, soit de 40 millisecondes pendant laquelle cette image doit être affichée à l'écran.
Si la lecture se fait dans de bonnes conditions, l'utilisateur ne remarquera pas de transition entre l'affichage d'images successives et aura ainsi l'impression de mouvement. Si le système est surchargé, l'application ne disposera plus des ressources suffisantes soit pour effectuer l'ensemble du décodage soit pour effectuer l'affichage. Cela se traduit par une réduction du nombre d'images affichées et une discontinuité dans le flux entraînant un affichage saccadé.
Le critère de détection du mode dégradé (voir section 4.5.1) est, dans le cas d'une lecture vidéo, le nombre d'images par seconde affichées par l'application.

5.2 Mécanismes mis en jeu

L'affichage graphique dépend non seulement du pilote de la carte vidéo mais aussi du gestionnaire graphique (serveur X), qui s'occupe de l'affichage vidéo dans un environnement de travail de type GNU/Linux. Le serveur X est une application comme une autre, donc dans le cas ou le système est surchargé, l'interaction entre les applications et l'utilisateur se dégrade, car tous les événements transmis par l'utilisateur doivent être interceptés par le serveur X puis transmis à l'application correspondante.
Il existe plusieurs interfaces permettant de gérer l'affichage vidéo dans une application. En utilisant une librairie de bas niveau, dans notre cas SDL (Simple Direct-Media Layer), il est possible d'accéder directement à la mémoire vidéo, c'est à dire d'outrepasser la couche du

FIG. 4 – L'interface utilisateur.

serveur X et donc d'isoler au mieux notre application, afin d'éviter que son exécution soit contrainte par celle d'un autre programme.

5.3 Évaluation de l'affichage

Une des fonctions utilisées dans l'application crée permet de mettre à jour les données affichées à l'écran. Cela consiste à mettre à jour la mémoire vidéo. Ainsi, à chaque nouvelle image qui a été décodée, le programme fait appel à cette fonction pour afficher celle-ci à l'écran. On considère donc les temps de passage dans cette fonction et calculons le nombre d'exécution. À chaque intervalle de temps d'au moins une seconde, est obtenue une valeur représentant le nombre d'images affichées pendant la dernière seconde écoulée. Il faut ensuite comparer ce résultat à la valeur seuil de qualité désiré et le cas échéant mettre en place la hiérarchie d'ordonnancement.

5.4 Exécution dans la hiérarchie

Une fois l'application migrée dans son propre ordonnanceur, elle ne partage plus équitablement le processeur avec les autres applications, mais dispose d'un taux d'allocation minimum. L'application est donc garantie de disposer des ressources nécessaires à son bon déroulement. Les autres applications continuent leur exécution normalement et se partagent équitablement le processeur entre elles.

5.5 Interface avec l'utilisateur

Une interface graphique (Voir figure 4) permet à l'utilisateur de pouvoir finement contrôler le niveau de ressources dédiés. Pour cela l'utilisateur renseigne l'identificateur du processus, le bouton "Manage" sert à créer la hiérarchie d'ordonnanceur et un bouton permet d'ajuster le pourcentage du taux processeur. Il peut donc régler le taux dynamiquement en fonction de la charge du système au cours de la session. Le bouton "Retire" enlève le processus de son ordonnanceur si celui-ci n'est pas déjà terminé et remet l'ordonnanceur dans l'état initial.

6 ÉVALUATION

6.1 Coût de la mise en place

Le mécanisme permettant de créer la hiérarchie se décompose en plusieurs étapes. Le temps nécessaire pour vérifier la présence des modules, charger ceux-ci dans le noyaux lorsqu'il ne sont pas déjà présents, les connecter dans la hiérarchie et affecter les taux correspondants a été évalué. L'ensemble de ces opérations s'effectue en moyenne, sur un jeu de vingt tests, en 225 millisecondes sur un ordinateur doté d'un processeur cadencé à 660 Mhz, ce qui est relativement court comparé à la durée d'exécution d'un programme. L'utilisateur ne sera donc pas gêné lors de la mise en place de la plate-forme.

6.2 Comparaison

Le système NEMESIS (voir section 2.1) repose sur l'attribution des ressources basé sur des durées d'utilisation. Cela dépend donc fortement de la machine et de l'application, car il faut spécifier pour chaque ressource mise en jeu une durée représentant le temps de réservation.
De même dans SMART (voir section 2.2), la première interface proposé utilisant des durées d'exécution de section de code ne permet pas d'affiner la répartition des ressource mais seulement de dégrader l'application. La seconde interface en revanche, destinée à l'utilisateur, permet de modifier dynamiquement la priorité de l'application. SMART n'offre donc pas la possiblité à l'application d'adapter automatiquement son taux d'allocation processeur.
Enfin, le système d'exploitation QLinux (voir section 2.1) permet aussi de s'affranchir du modéle d'exécution basé sur les durées d'utilisation des ressources.

7 CONCLUSION

Nous avons présenté un système capable de répondre au problème de la qualité de service dans les systèmes actuels en modifiant la politique d'allocation du processeur en fonction des besoins de l'utilisateur. Notre approche se base sur la priorité mise en œuvre par une hiérarchie d'ordonnanceur, et permet donc de garder le modèle de processus aussi simple que possible, sans avoir à utiliser de données temporelles difficiles à obtenir sur le modèle d'exécution des applications, mais un paramètre reflétant le niveau d'exécution.
Ainsi, en utilisant conjointement la Machine Virtuelle Virtuelle pour construire l'environnement d'exécution dédié et Bossa, permettant de bénéficier d'un framework pour charger et décharger les ordonnanceurs, on est parvenu à réaliser une plate-forme dans laquelle l'utilisateur peut spécifier explicitement l'importance d'une application, modifier la politique du système et garantir la qualité de service nécessaire.
Les besoins des applications, comme pour les serveurs internet, la bande passante et le nombre de requêtes qu'ils reçoivent, sont amenés à évoluer au cours du temps. L'adaptation des politiques de gestion de ressources en fonction de ces paramètres permet aux applications de servir les utilisateurs dans les meilleures conditions, sans avoir à modifier le logiciel, car c'est l'environnement d'exécution est mis à jour dynamiquement.
D'une manière plus large, il serait intéressant de travailler sur l'adaptation d'autres politiques de partage de ressources au niveau applicatif, ainsi que de mettre en œuvre un serveur centralisé responsable de la qualité de service pour les applications multimédias.

BIBLIOGRAPHIE

[Barreto, 2001] Barreto L., Bossa : A DSL Framework for Application-Specific Scheduling Policies. Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, IEEE Computer Society, p. 161 (2001).

[Barreto, 2002] Barreto L., Douence R., Muller G., Sudholt M., Programming os schedulers with domain-specific languages and aspects : New approaches for os kernel engineering. Proceedings of the 1st AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software, Apr. (2002).

[Etsion, 2004] Etsion Y., Tsafrir D., G. Feitelson D., Desktop scheduling : how can we know what the user wants ?. NOSSDAV '04 : Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video, ACM Press, p. 110-115, Cork, Ireland (2004).

[Folliot, 1998] Folliot B., Piumarta I., Riccardi F., A Dynamically Configurable, Multi-Language Execution Platform. 8th ACM SIGOPS European Workshop, p. 175-181, Sintra, Portugal (1998).

[Leslie, 1996] Leslie I., McAuley D., Black R., Roscoe T., Barham P., Evers D., Fairbairns R., Hyden E., The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. IEEE Journal of Selected Areas in Communications, Vol. 14, N°7, p. 1280-1297 (1996).

[Nieh, 2003] Nieh J., Lam M., A SMART scheduler for multimedia applications. ACM Transactions on Computer Systems (TOCS), ACM Press, Vol. 21, N°2, p. 117-163 (2003).

[Piumarta, 2000] Piumarta I., Folliot B., Seinturier L., Carine Baillarguet, Highly configurable operating systems : the VVM approach. ECOOP'2000 Workshop on Object Orientation and Operating Systems, Cannes, France (2000).

[Reed, 1997] Reed D., Fairbairns R., The Nemesis Kernel Overview (1997).

[Regehr, 2001] Regehr J., Using Hierarchical Scheduling to Support Soft Real-Time Applications on General-Purpose Operating Systems. PhD thesis, University of Virginia (2001).

[Sundaram, 2000] Sundaram V., Chandra A., Goyal P., Shenoy P., Sahni J., Vin H., Application performance in the QLinux multimedia operating system. Proceedings of the eighth ACM international conference on Multimedia, ACM Press, p. 127-136, Marina del Rey, California, United States (2000).