MASTER EN SCIENCES INFORMATIQUES

Principes de gestion dynamique des systèmes à mémoire virtuelle

Trân-Quôc-Tê

Award date:
1977

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES

N-D DE LA PAIX

NAMUR

INSTITUT D'INFORMATIQUE

PRINCIPES DE GESTION DYNAMIQUE DES SYSTEMES A MEMOIRE VIRTUELLE

Thèse présentée par Trân-Quôc-Tê
pour l'obtention du titre de
docteur en sciences (option informatique)
Décembre 1977

Jury :

M. Noirhomme, promoteur,
C. Cherton,
P.J. Courtois,
J. Fichefet,
H. Leroy, membres.

Je tiens d'abord à englober tous les membres de l'Institut d'Informatique dans mes remerciements pour toute l'aide matérielle, logicielle, et spirituelle, qu'ils m'ont prodiguée tout au long de ce travail ;

plus particulièrement ceux des groupes "Mesures et Performances" et "Mathématiques Appliquées" : Madame Noirhomme, Messieurs Fichefet, Leclercq, et Ramakaers, pour leur encadrement et les discussions fructueuses que j'ai eues avec eux ;

Messieurs Cherton et Leroy, pour leurs remarques nombreuses et toujours pertinentes qui m'ont conduit à approfondir beaucoup d'aspects de cette thèse.

Que Monsieur Courtois, qui m'a aidé à avoir une vue plus exacte du concept de "localité", trouve ici l'expression de ma sincère gratitude.

Je n'oublierai pas Monsieur Gelenbe dont tous les conseils, et toute la gentillesse, ont été pour moi un encouragement continuel.

TABLE DES MATIERES

Chapitre 3 : **Algorithmes de pagination à partition fixe**

Chapitre 4 : Allocation à partition variable

Chapitre 5 : Aspects du contrôle global des systèmes à mémoire virtuelle

1. LE CONTEXTE DE LA THESE

1.1. Systèmes de multiprogrammation.

En des termes tout à fait généraux, un système informatique peut être décrit comme un certain ensemble structuré de ressources. Chacune de ces ressources est spécifiquement conçue et/ou dédiée à une fonction particulière. Un programme est un processus demandeur de ressources : son exécution est constituée d'une séquence d'opérations élémentaires, chacune desquelles requérant l'utilisation de certaine(s) ressource(s) pendant une certaine durée de temps.

Parmi ces ressources, les processeurs sont les unités qui "traitent" les informations. Chaque traitement élémentaire (ou instruction) nécessite un accès à certaines informations, comme le code de l'instruction et les opérandes. On utilise le terme générique de mémoire pour désigner les supports physiques sur lesquels sont stockées les informations.

A proprement parler, il n'existe pas une seule mémoire, mais plutôt plusieurs types, ou niveaux, de mémoire. Les informations ne sont accessibles du processeur qu'à partir de niveaux appelés, pour cette raison, centraux. La mémoire centrale, coûteuse pour des raisons technologiques, est, par ce fait même, relativement limitée. Elle est alors prolongée par des mémoires périphériques ou secondaires, à la fois beaucoup plus vastes et moins coûteuses. Une telle hiérarchie de mémoires est généralement utilisée de la façon suivante : la mémoire centrale sert au

stockage du code objet des programmes, tandis que les informations d'un usage occasionnel (fichiers de données, par exemple) sont mises sur les niveaux périphériques.

Afin de permettre une utilisation rationnelle et efficace des différentes ressources, un tel système travaille généralement en multiprogrammation. Rappelons que ce terme désigne un mode de fonctionnement selon lequel plusieurs programmes peuvent être simultanément pris en charge par le système. Une ressource momentanément non utilisée par un programme peut alors être allouée à un autre programme. Cette utilisation en parallèle des ressources améliore leurs taux d'activité individuels, et, par conséquent, les performances du système tout entier.

La gestion d'un système de multiprogrammation pose des problèmes décisionnels complexes dont la plupart sont loin d'être maîtrisés. D'une façon informelle, il s'agit d'allouer les ressources aux programmes de manière à optimiser certaines mesures de la performance du système.

Un des problèmes clés est l'allocation de la mémoire centrale. D'une part, étant elle-même limitée, la mémoire centrale limite la taille et le nombre de programmes pouvant être exécutés simultanément; elle restreint donc la capacité de multiprogrammation du système. D'autre part, les performances propres à l'allocation de la mémoire conditionnent étroitement celles globales du système.

Parmi les techniques permettant un usage partagé de la mémoire centrale, celle de la mémoire virtuelle est de plus en

plus utilisée dans les systèmes actuels. Le paragraphe suivant en décrit les principes de base.

1.2. Mémoire virtuelle.

Cette technique doit son nom au fait qu'elle permet de reculer, "virtuellement", les limites physiques de la mémoire centrale. Contrairement aux méthodes d'allocation selon lesquelles le code objet des programmes est chargé entièrement en mémoire centrale, et y est maintenue statiquement jusqu'à la fin, il s'agit d'une méthode d'allocation qu'on peut qualifier de partielle et dynamique.

La raison d'être de la mémoire virtuelle est essentiellement la suivante. Au cours de l'exécution d'un programme, les informations accédées sont généralement concentrées dans certaines zones spécifiques du code objet. Du fait de cette localisation des références, les besoins instantanés en espace des programmes sont assez réduits, comparativement à la taille totale de leur code objet. Pour chacun des programmes, il est donc raisonnable de n'en garder en mémoire centrale que les parties momentanément requises. Les parties restantes sont mises sur une mémoire secondaire et chargées en mémoire centrale seulement en fonction des besoins.

Ce mode d'utilisation de la mémoire peut être sous la responsabilité du programmeur : celui-ci découpe lui-même son programme et spécifie au système les moments opportuns pour charger ou décharger telles ou telles parties; c'est la méthode dite des overlays. Au contraire, la mémoire virtuelle réfère au cas

où cette allocation partielle et dynamique de la mémoire est entièrement à la charge du système.

Le découpage des programmes se fait alors au moment de la compilation. Leur code objet est partitionné en blocs contigus appelés segments, ou pages lorsqu'ils sont tous de même taille. Chaque information est identifiée par une adresse virtuelle de la forme : (identité du segment, déplacement à partir du début du segment). Les transferts d'informations entre la mémoire centrale et son prolongement virtuel se feront toujours par segment entier, d'un seul tenant. Pour convertir une adresse virtuelle en adresse physique (emplacement actuel d'une information), il suffit alors d'une table des segments, comme il est illustré à la figure 1.

Tout au long de l'exécution des programmes, le système doit également décider de la répartition dynamique des segments d'informations, et par conséquent de leurs mouvements, entre les différents niveaux de la mémoire. Les segments requis par le processeur doivent être présents en mémoire centrale. Par conséquent, lorsqu'un programme encourt un défaut de segment (référence à un segment absent de la mémoire centrale), il est interrompu au moins jusqu'à ce que le segment fautif soit chargé en mémoire centrale. Les décisions d'allocation sont donc une fonction cruciale : chaque programme devrait être interrompu le moins souvent possible par ces attentes de transferts.

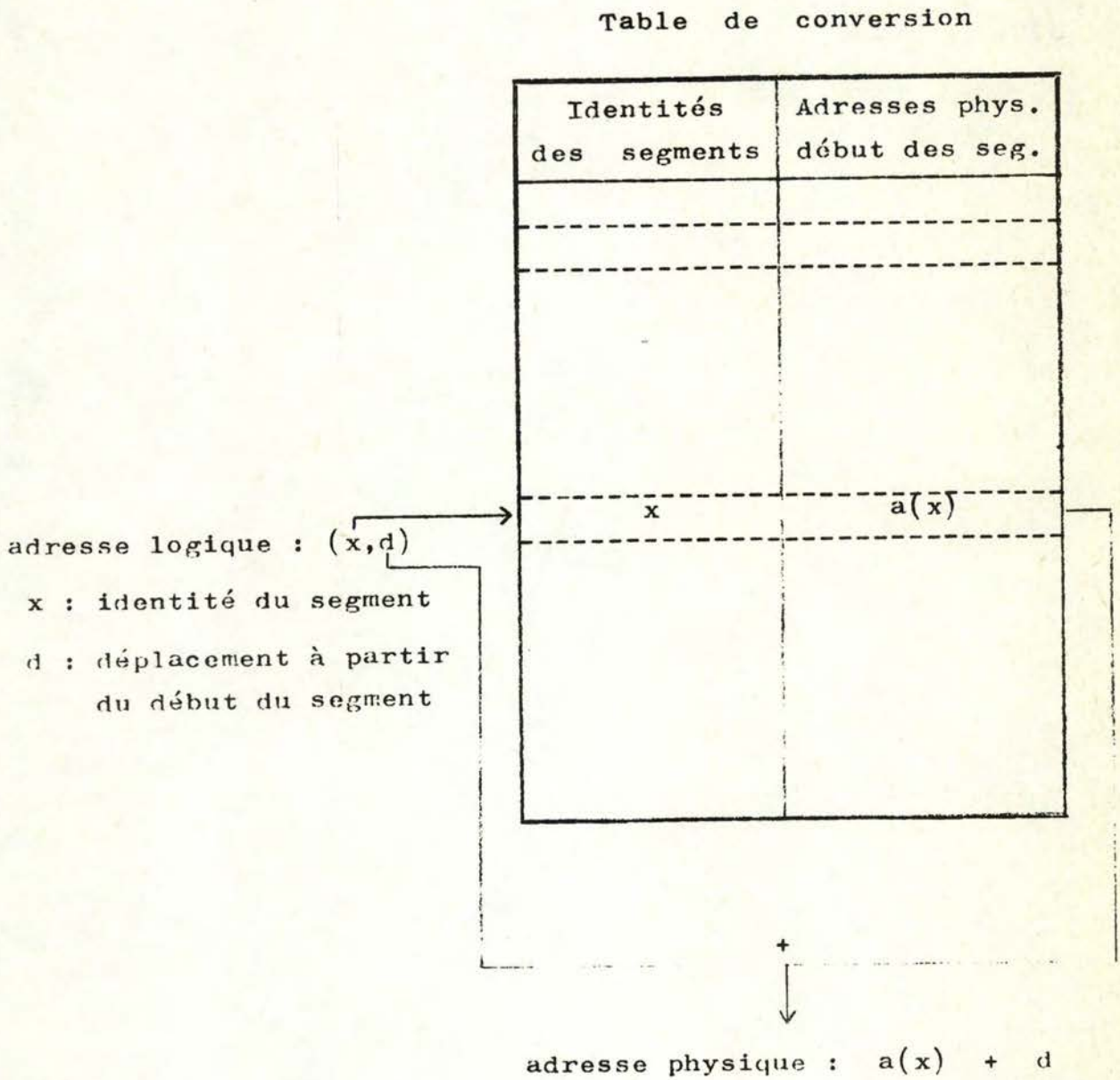


Fig. 1. Conversion d'une adresse logique en adresse physique.

1.3. Plan de la thèse.

L'objet de cette thèse est l'analyse des problèmes de contrôle et de décision qui se posent dans la gestion dynamique d'un système à mémoire virtuelle paginée. Grosso modo, ces problèmes se situent à 2 niveaux :

- d'une part, l'allocation de l'espace en mémoire centrale à chacun des programmes, considéré individuellement, et
- d'autre part, l'allocation des ressources du réseau à un ensemble de programmes, c'est-à-dire les interactions entre les différents programmes au niveau global du système.

Le premier problème sera essentiellement formulé comme un problème d'allocation dynamique d'une mémoire centrale, considérée comme un ensemble de cadres de page disponibles, à des programmes, considérés comme des processus générateurs de demandes de pages. Ce problème de décision est en avenir non déterministe : pratiquement, les références des programmes ne sont jamais connues à l'avance. Toute stratégie d'allocation de la mémoire repose donc sur une méthode de prévision : au fur et à mesure du déroulement d'un programme, son comportement passé est mémorisé pour estimer celui à venir, conformément à certaines idées que l'on a sur les propriétés corrélatives de ce comportement. Intuitivement, l'incidence de ces estimations dynamiques et instantanées sur chacune des décisions élémentaires d'allocation devrait être la suivante : les pages estimées comme probablement requises dans un avenir proche sont maintenues (ou chargées) en mémoire centrale, tandis que celles peu probables sont laissées (ou déchargées) sur la

mémoire secondaire. Un thème central de cette thèse est précisément de justifier l'efficacité de ce principe heuristique.

Préalablement à l'analyse des processus d'allocation d'espace, le chapitre 2 étudie les processus des demandes de pages générées par un programme au cours de son exécution. Les modèles de prévision du comportement sont formulées comme des règles d'estimation des références futures sur la base de celles passées. Ensuite, nous discuterons de la modélisation des programmes de façon à refléter leur tendance à la localisation. Nous terminerons ce chapitre par des exemples de modèles les plus utilisés.

Le chapitre 3 est consacré aux algorithmes de pagination à partition fixe. Sous ce mode d'allocation, chaque programme ne peut jamais occuper qu'une zone de taille fixe de la mémoire centrale. Lorsqu'une page doit être chargée alors que le programme occupe déjà le nombre maximum de cadres qui lui est alloué, il faut choisir une page du programme à remplacer de la mémoire. Pour des programmes localisés, nous justifierons le remplacement de la page la moins susceptible d'être requise à la référence suivante. Les performances de ce critère "à terme immédiat" seront également analysées à travers quelques modèles de comportement originaux, inspirés des méthodes dites de lissage utilisées dans la prévision des séries chronologiques.

Les stratégies d'allocation à partition variable, sous lesquelles aucune limite fixe n'est imposée quant à la quantité d'espace allouée à chacun des programmes, font l'objet du cha-

pitre 4. Les stratégies d'allocation devraient alors réaliser un certain compromis entre la minimisation du nombre de défauts de page et la minimisation de l'espace alloué au programme. En nous inspirant du principe de l'offre et de la demande, nous énoncerons une formulation générale des stratégies à l'ensemble d'activité (working set) : pour chaque instruction élémentaire, on n'allouera que les pages dont la probabilité d'être référencées dans l'immédiat excède un certain niveau fixé a priori. Ce principe d'allocation sera particularisé à divers modèles de comportement, et justifié dans le cas des comportements localisés.

Le chapitre 5 présente l'analyse de quelques problèmes de contrôle d'un système de multiprogrammation à mémoire virtuelle. D'une façon informelle, étant donné un certain ensemble de programmes à exécuter, il s'agit de déterminer les règles de partage de la mémoire centrale et les règles de priorité à chacune des unités du système, cela de façon à ce que certaines unités fondamentales du système, les processeurs et les entrées-sorties par exemple, soient utilisées le plus possible.

Une première question se pose à propos de la charge optimale à imposer au système. Une approche classique, basée sur la théorie des réseaux d'attente, permet de mettre en évidence l'existence d'un degré de multiprogrammation (nombre de programmes dans le système) optimal. D'autres approches, plus expérimentales, préconisent une régularisation automatique de la charge basée sur l'utilisation de l'unité de pagination (Leroudier et Potier [LeP76]) ou sur la durée moyenne de temps virtuel entre 2 défauts

de page (Denning et Kahn /DeK76/) : l'unité de pagination trop souvent sollicitée, ou des défauts de page trop rapprochés, constituent souvent des critères indiquant une surcharge possible du système, tous les programmes s'étranglant au niveau de l'unité de pagination (thrashing). La caractérisation de la charge optimale reste un problème ouvert, au stade des recherches actuelles.

Le problème du partage de la mémoire centrale, assez complexe, sera illustré dans le cas de 2 programmes. Dans ce cas, la partition optimale peut être déterminée par une simple procédure récursive. Cette procédure conduit généralement à des partitions biaisées, c'est-à-dire favorisant fortement un programme au détriment de l'autre.

La coordination temporelle des programmes à l'intérieur du système sera finalement traitée comme un problème de programmation dynamique. Une conclusion importante est alors qu'il faut accorder la priorité aux programmes les moins sujets à des défauts de page.

2. MODELISATION DU COMPORTEMENT DE PROGRAMME — LOCALISATION DES REFERENCES

2.1. Présentation du chapitre.

Ce chapitre est consacré à l'étude du comportement de programme. Par cette expression, on entend toutes les propriétés de la séquence des pages référencées par un programme tout au long de son exécution. Ces propriétés, intrinsèques au programme considéré, ne servent pas seulement à une description statique du programme. La connaissance de la façon dont les programmes se comportent en général est un préalable indispensable à la conception des règles efficaces d'allocation de la mémoire.

Plutôt qu'un ensemble de postulats probabilistes sur un certain processus aléatoire, nous considérerons essentiellemt un modèle de comportement de programme comme une méthode adaptative de prévision de ce comportement. Une telle méthode est donnée sous une forme explicite lorsqu'elle fournit des estimations d'une façon précise, et sous une forme implicite quand elle génère seulement un ordre sur l'ensemble des pages.

Une attention particulière sera accordée à la modélisation des programmes de façon à refléter une tendance communément observée, la localisation des références. Nous présenterons des conditions de localisation à laquelle satisfont la plupart des modèles considérés dans la littérature: lru, lfu,...

Nous présenterons enfin un test général permettant de valider des modèles sur la base d'un comportement observé.

2.2. Formulation des méthodes de prévision du comportement.

Le but d'une stratégie d'allocation de la mémoire centrale est de satisfaire, "au mieux", les besoins dynamiques en pages des programmes. Elle nécessite donc, de la part du système, une certaine connaissance de ces besoins. Considérons une exécution particulière d'un programme déterminé ; soit alors

$$s = r_1.r_2...r_t...$$

la séquence des pages successivement référencées (r_t désigne la page contenant les informations requises pour la $t^{ième}$ opération élémentaire). Cette séquence reflète l'écriture du programme et la façon dont les pages sont découpées. Néanmoins, elle peut différer d'une exécution à l'autre, les "données" à traiter n'étant pas toujours les mêmes. Considérons la population idéalisée constituée de toutes les séquences particulières générées par tous les passages possibles du même programme ; soit Pr la distribution de probabilité qui caractérise cette population. Une séquence spécifique s peut alors être considérée comme une réalisation particulière d'un processus stochastique

$$S = R_1.R_2...R_t...$$

à temps discret (seul l'ordre dans lequel les pages sont référencées nous intéresse), et à états également discrets (chaque référence est un élément de l'ensemble fini X de toutes les pages du programme).

Cette distribution de probabilité, intrinsèque au programme, peut être estimée à partir d'un grand nombre d'exécutions antérieures du programme. Pour chacune d'elles, la trace des pages est relevée, ce qui permet alors d'estimer certains paramètres

statistiques importants. Cependant, une telle méthode est évidemment très lourde à supporter par le système, et ne saurait se justifier que pour certains programmes d'un usage très fréquent.

En fait, puisque chaque décision élémentaire d'allocation n'est prise qu'au fur et à mesure du déroulement du programme, il ne s'agit pas tellement de connaître les références globalement et préalablement à l'exécution, mais surtout de disposer d'une méthode adaptative de prévision du comportement. Nous nous limiterons dans cette thèse à l'étude de ce genre de méthodes. Nous appellerons une telle méthode un modèle de prévision du comportement. Un modèle est donné sous une forme explicite quand il permet de quantifier les estimations d'occurrence des pages. Sous une forme implicite, il fournit simplement un ordre de préférence sur les différentes pages du programme.

2.2.1. Modèles explicites.

Ce sont des méthodes adaptatives qui, sur la base d'information relatives au comportement passé, génèrent des prévisions de références pour chacune des pages. Plus formellement, appelons :

X l'ensemble des pages du programme,

état passé tout couple (t, s_{t-1}), avec $t = 1,2,\ldots$ et $s_{t-1} \in X^{t-1}$,

Q l'espace des états passés,

vecteur de prévision tout ensemble de réels $\{a(x);\ x \in X\}$ tel que

$$a(x) \in [0, 1]\ ,\ \forall\ x \in X \quad \text{et} \quad \sum_{x \in X} a(x) = 1,$$

A l'espace des vecteurs de prévisions.

Définition 2.1.

Un modèle explicite est une application :

$$p : Q \longrightarrow A : (t, s_{t-1}) \longrightarrow p_t(., s_{t-1}).$$

Nous appellerons $p_t(x, s_{t-1})$ la prévision de référence de la page x à l'instant t, étant donné le passé s_{t-1}.

Un modèle p sera dit vrai pour un programme si ses prévisions concordent avec le comportement intrinsèque du programme. Plus précisément :

Définition 2.2.

Nous dirons qu'un modèle p est vrai pour une chaîne S, ou encore que S obéit à p, si, quels que soient t, s_{t-1}, et x :

$$p_t(x, s_{t-1}) = Pr[R_t=x \mid S_{t-1}=s_{t-1}]$$

($S_{t-1} = R_1 \ldots R_{t-1}$ désigne la séquence des t-1 premiers éléments de S).

2.2.2. Modèles implicites ou de rangement.

Un modèle de prévision du comportement est sous une forme implicite lorsqu'il ne spécifie pas les prévisions de référence d'une façon précise, mais fournit seulement l'ordre de grandeur relatif de ces prévisions. Bien que moins riche que la formulation précédente, une formulation implicite permet quand-même la validation des modèles (§ 2.4), et surtout la conception et l'analyse d'un grand nombre d'algorithmes de pagination (chap. 3).

Appelons tout d'abord :

$n = |X|$ le nombre de pages du programme,

rangement de X toute bijection $b : X \longrightarrow \{1,\ldots,n\}$,

B l'ensemble de tous les rangements de X.

Nous avons alors la

Définition 2.3.

> Un modèle implicite ou de rangement est une application :
>
> $m : Q \longrightarrow B : (t, s_{t-1}) \longrightarrow m_t(., s_{t-1})$.
>
> Pour toute page x et pour tout état passé (t, s_{t-1}), nous appellerons $m_t(x, s_{t-1})$ le rang de x à l'instant t étant donné le passé s_{t-1}.

Il faut maintenant spécifier le "critère" selon lequel les pages sont rangées. Nous adopterons la

Définition 2.4.

> Nous dirons qu'un modèle implicite m est vrai pour une chaîne S, ou encore que S obéit à m, si, pour tout couple de pages x et y et pour tout état passé (t, s_{t-1}), nous avons l'implication :
>
> $$m_t(x, s_{t-1}) \succ m_t(y, s_{t-1}) \Rightarrow \Pr[R_t=x \mid S_{t-1}=s_{t-1}] \leq \Pr[R_t=y \mid S_{t-1}=s_{t-1}]$$

Un modèle implicite vrai range donc les pages dans l'ordre décroissant de leur probabilité d'occurrence à la prochaine étape, la page la plus probable recevant le rang 1, ..., celle la moins probable, le rang n.

Supposons maintenant que nous disposons d'un modèle explicite. Alors, en ordonnant les pages selon leur prévision de référence, nous obtenons un modèle implicite, dit induit par le modèle explicite original. D'une façon plus précise :

Définition 2.5.

> Un modèle implicite m est induit par un modèle explicite p si, pour tout couple de pages x et y et pour tout état passé

(t, s_{t-1}), nous avons l'implication :

$$m_t(x, s_{t-1}) \succ m_t(y, s_{t-1}) \Rightarrow p_t(x, s_{t-1}) \preceq p_t(y, s_{t-1}).$$

Avec ces définitions, il est évident que tout modèle de rangement induit par un modèle explicite vrai est également vrai.

Remarques.

- Des modèles explicites différents peuvent induire un même modèle de rangement, pourvu que les ordres induits par les prévisions de référence des pages soient constamment les mêmes.
- Un modèle explicite n'induit pas toujours un modèle de rangement d'une façon univoque. Nous pouvons convenir de départager deux pages de prévisions égales en accordant le rang le plus élevé à la page dont le rang à l'étape précédente était le plus plus élevé (un rangement initial de X est supposé donné).
- A partir d'un modèle explicite p, au lieu d'ordonner les pages suivant leur prévision pour la prochaine étape, on peut envisager d'autres critères de rangement. Notamment, le délai moyen de récurrence des pages peut être déterminé de la façon suivante.

Pour toute page x et pour tout état passé (t, s_{t-1}), définissons :

$$q_t^{(1)}(x, s_{t-1}) = 1 - p_t(x, s_{t-1}), \qquad (2.1)$$

et, d'une façon récursive, pour tout entier h supérieur à 1 :

$$q_t^{(h)}(x, s_{t-1}) = \sum_{r \neq x} p_t(r, s_{t-1}) . q_{t+1}^{(h-1)}(x, s_{t-1}.r). \qquad (2.2)$$

Si le modèle p est vrai, alors $q_t^{(h)}(x, s_{t-1})$ est la probabilité pour que x ne soit pas référencé avant h étapes :

$$q_t^{(h)}(x, s_{t-1}) = \Pr\left[R_{t-1+j} \neq x, \ \forall\ j=1,\ldots,h \mid S_{t-1}=s_{t-1}\right], \qquad (2.3)$$

de sorte que, dans ce cas, le délai moyen de récurrence de x vaut :

$$\bar{p}_t(x, s_{t-1}) = \sum_{h=1}^{\infty} q_t^{(h)}(x, s_{t-1}). \qquad (2.4)$$

2.3. Modèles reflétant la localisation.

Après ces définitions assez formelles des modèles de prévision du comportement, une question fondamentale se pose en ce qui concerne le choix d'un "bon" modèle. Un modèle réaliste permet tout d'abord d'analyser les processus de demande et d'allocation d'espace avec un maximum de vraisemblance. Mais surtout, d'un point de vue plus pratique, chaque décision instantanée d'allocation repose essentiellement sur des estimations du comportement à venir des programmes. Les performances du processus d'allocation exigent donc que ces estimations soient les plus fiables possibles.

En plus de son adéquation à des comportements réels, la méthode de prévision doit aussi être aussi simple que possible. En effet, une méthode qui exige la mémorisation et la mise à jour d'un trop grand nombre d'informations, et/ou des procédures de calcul trop complexes, risque, par ces charges qu'elle impose au système, d'en détériorer les performances. La procédure de prévision doit encore être autonome et adaptative : elle ne peut se baser que sur le comportement passé observé des programmes pour estimer, au fur et à mesure, leur comportement futur.

Le comportement des programmes a fait l'objet de nombreuses études expérimentales. Toutes ces études confirment, à des degrés divers, une caractéristique commune à la plupart des programmes, à savoir leur tendance à la localisation.

2.3.1. Description de la localisation.

Lorsqu'on suit l'exécution d'un programme au cours du temps, on constate généralement que les informations accédées ne s'éparpillent pas au hasard, ni de façon uniforme, dans l'espace des adresses. Au contraire, cette exécution est constituée d'une séquence de "phases" plus ou moins longues, au cours de chacune desquelles les références sont fortement concentrées dans certaines parties spécifiques du code objet [BrG68, Den68a-b, BeK69, Say69, Hat72, LeB75, MaB76, BaM76, CoV76].

Cette propriété du comportement, connue sous le nom de localisation des références, peut s'interpréter en termes de "logique" des programmes. Chacune de ces phases correspond à l'exécution d'une certaine unité logique de calcul, routine, ou procédure du programme. Une telle procédure comporte généralement de nombreuses itérations sur certaines séquences d'instructions spécifiques, et accède à des données généralement voisines (éléments consécutifs d'un tableau, par exemple).

A cause du quasi-déterminisme dans certaines boucles, les affinités séquentielles entre pages sont souvent l'un ou l'autre de ces extrêmes : certaines pages sont fort liées logiquement entre elles, tandis que d'autres s'excluent presque tout à fait. La matrice des fréquences de transition inter-pages est donc presque complètement décomposable, et cette propriété a permis une analyse approchée de certains modèles markoviens [CoV76].

2.3.2. Conditions explicite et implicite de localisation.

Après cette description du phénomène de la localisation des références, il s'agit maintenant de concevoir des méthodes adaptatives permettant de détecter, pour chacune des phases d'un programme, ses pages momentanément privilégiées. La plupart des méthodes utilisées (lru, lfu, etc...; voir § 2.3.3) reposent sur les considérations suivantes.

Supposons que nous soyons au milieu d'une phase de localisation, caractérisée notamment par un ensemble de pages privilégiées. Comment nous y prendrions-nous pour déterminer cet ensemble, sur la base du comportement passé du programme ? Les pages les plus susceptibles d'une grande utilisation jusqu'à la fin de la phase sont précisément celles-là mêmes les plus souvent référencées depuis le début de cette phase. Ceci nous amène à considérer le "principe" de prévision suivant : les pages fréquemment utilisées (resp. rarement sollicitées) dans un passé récent sont prévues comme probablement réutilisées (resp. vraisemblablement non requises) dans un futur proche.

Dans le cadre d'un modèle explicite du comportement, la prévision instantanée de référence d'une page devrait donc être une certaine mesure de l'utilisation récente de cette page. En d'autres termes, l'effet d'une référence doit être d'augmenter la prévision de la page référencée, et de diminuer celle des autres pages. Ces considérations sont formalisées par la

Définition 2.6.

| Un modèle explicite p satisfait à la condition explicite de localisation (CEL) si, quels que soient $(t, s_{t-1}) \in Q$, et r,

$x \in X$:

$$x \neq r \Rightarrow p_{t+1}(x, s_{t-1}.r) \leq p_t(x, s_{t-1}). \qquad (2.5)$$

Notons que, puisque la somme des prévisions doit valoir 1, celle de la page référencée r est non décroissante.

Une adaptation de la CEL au niveau des rangements conduit à la Définition 2.7.

Un modèle de rangement m satisfait à la condition implicite de localisation (CIL) si, quels que soient $(t, s_{t-1}) \in Q$, et $r, x \in X$:

$$x \neq r \Rightarrow m_{t+1}(x, s_{t-1}.r) \geq m_t(x, s_{t-1}). \qquad (2.6)$$

Remarquons que ces conditions, bien que similaires, ne sont pas équivalentes.

Ce genre de modèles vise essentiellement la détection adaptative des pages localement privilégiées, et n'a pas la prétention de refléter, le plus fidèlement possible, le comportement intrinsèque des programmes. Par exemple, après qu'un programme a itéré un grand nombre de fois dans une certaine boucle, les pages de la boucle posséderont bien des prévisions instantanées plus élevées que celles restantes ; néanmoins, parmi les pages privilégiées, ce sont bien au contraire celles qui viennent d'être référencées qui seront, ipso facto, les dernières à l'être de nouveau.

Ces méthodes peuvent encore être prises en défaut au début de chaque phase de localisation ou au cours d'une période de transition. Des études expérimentales récentes /MaB76, BaM76/ révèlent que ces changements peuvent être fort brusques, mais que les périodes transitoires ne constituent qu'une petite fraction de la vie d'un programme.

Pour une description plus fidèle et plus globale du comportement, on pourrait imbriquer différents modèles à court terme, du genre de ceux cités en exemple au § 2.3.3, dans des modèles de transition inter-phase. Ce genre d'approche n'a pu être analysée d'une façon satisfaisante que sous certaines hypothèses de presque décomposabilité /CoV76/. En tant que méthode de prévision, un tel modèle est évidemment plus lourd, et exige l'estimation préalable de plus de paramètres, que des modèles CEL ou CIL.

2.3.3. Exemples de modèles de comportement.

1) Le modèle à références indépendantes /ADU71/.

Il s'agit tout simplement d'un modèle explicite pour lequel les prévisions de référence de chacune des pages restent invariantes, quel que soit le comportement passé du programme :

$$p_t(x, s_{t-1}) = a(x), \forall x \text{ et } (t, s_{t-1}), \qquad (2.7)$$

où a(.) désigne un vecteur de prévision fixé. Ce modèle doit son nom au fait que, pour les chaînes obéissant à un tel modèle, les références successives sont indépendantes entre elles (et ont toutes la même distribution). Basé sur des estimations statiques, ce mode de prévision ne permet pas de détecter les sauts brusques de phase ; il peut raisonnablement décrire le comportement à l'intérieur d'une phase /CoV76/. En raison de sa relative simplicité analytique, ce modèle est l'un des plus étudiés.

2) Le modèle lru /NGST70, ShT72/.

Nous appellerons modèle lru (least recently used) tout modèle de rangement qui classe les pages suivant la chronologie passée de leur occurrence, la page utilisée la plus récemment recevant le rang 1, celle la moins récemment, le rang n. D'une façon plus précise, les rangements d'un tel modèle satisfont à la condition suivante :

quels que soient t, $s_{t-1} = r_1 \ldots r_{t-1} \in X^{t-1}$, et x, y ∈ X :

si $\max\{u \mid u \leq t-1 \text{ et } r_u = x\} > \max\{u \mid u \leq t-1 \text{ et } r_u = y\}$, (2.8)

alors $m_t(x, s_{t-1}) < m_t(y, s_{t-1})$.

Le rangement $m_t(., s_{t-1})$ est souvent appelé la pile lru (étant donné le passé (t, s_{t-1})), et $m_t(x, s_{t-1})$, la distance ou position de x dans la pile. La procédure de mise à jour de la pile lru a été décrite en détail dans [MGST70] : à chaque référence, la page référencée est amenée en haut de la pile (position 1), et les autres pages conservent leur rang relatif. Contrairement au modèle à références indépendantes, le modèle lru est très adaptatif, et peut convenir pour la prévision des chaînes fortement localisées.

Un modèle explicite pouvant induire le modèle lru est le modèle à liste lru (lru-stack model) [ShT72] : quels que soient t, s_{t-1}, et x :

$$p_t(x, s_{t-1}) = b_{m_t(x, s_{t-1})}, \tag{2.9}$$

où $\{b_i ; i = 1, \ldots, n\}$ est un vecteur de probabilité fixé, et où $m_t(x, s_{t-1})$ est la position de x dans la pile lru. Pour des chaînes obéissant à un tel modèle, les positions dans la pile des pages consécutivement référencées sont donc indépendantes entre elles et identiquement distribuées. Afin de tenir compte de la localisation, on considère souvent les b_i non croissants :

$$b_i \geq b_{i+1}, \quad i = 1, \ldots, n-1. \tag{2.10}$$

Dans ce cas, le modèle induit est trivialement un modèle lru. Dans la pratique, ce genre de modèles semble assez satisfaisant pour un grand nombre de comportements réels [SpD72, LeB75].

3) Le modèle lfu.

Les modèles lru de l'exemple précédent, qui supposent que les pages utilisées les plus récemment sont toujours celles les plus probablement réutilisées, peuvent surestimer la localisation des programmes. Un premier exemple de modèles qui accordent moins de "poids" aux références récentes peut être fourni par le modèle lfu. Il s'agit d'un modèle explicite pour lequel (à partir d'un certain instant) les prévisions de référence des pages ne sont autres que les fréquences relatives de leur occurrence passée. En d'autres termes, pour toute réalisation particulière $s_{t-1} = r_1 \ldots r_{t-1}$, nous avons :

$$p_t(., s_{t-1}) = \frac{1}{t-1} \sum_{u=1}^{t-1} d(., r_u), \qquad (2.11)$$

où nous avons posé :

$$d(x, y) = \begin{cases} 1, & \text{si les pages x et y sont les mêmes,} \\ 0, & \text{dans le cas contraire.} \end{cases} \qquad (2.12)$$

Signalons la relation suivante entre des prévisions consécutives :

$$p_{t+1}(., s_{t-1}.r) = \frac{t-1}{t} p_t(., s_{t-1}) + \frac{1}{t} d(., r) \qquad (2.13)$$

Des généralisations de ces modèles triviaux, satisfaisant à la CEL et/ou à la CIL seront analysées plus en détail dans les chapitres suivants.

2.4. Un test non paramétrique pour la validation des modèles.

Nous terminerons ce chapitre par un résultat concernant la validation des modèles de comportement, considérés comme des hypothèses statistiques sur des propriétés corrélatives des chaînes de références. Le test de validation présenté est non paramétrique, en ce sens qu'il est applicable à tout modèle, même donné sous une forme implicite. Son principe est tout à fait trivial : puisqu'un modèle m vrai range les pages par ordre décroissant de leur probabilité d'occurrence, on doit alors s'attendre à ce que les rangs $m_t(R_t, S_{t-1})$ soient constamment assez peu élevés. Le test consistera tout simplement à calculer les rangs moyens observés

$$\overline{m}_T = \frac{1}{T} \sum_{t=1}^{T} m_t(R_t, S_{t-1}), \qquad (2.14)$$

qui mesurent l'inadéquation du modèle m à la chaîne observée S, et à rejeter m si $\overline{m}_T$ excède (n+1)/2, n désignant le nombre total de pages du programmes. Nous montrerons en effet que l'erreur de première espèce, c'est-à-dire la probabilité de rejeter un modèle vrai, tend vers zéro lorsque T tend vers l'infini.

Lemme 2.1.

Soit $p = \{p_i\}$ un vecteur de probabilité sur $N = \{1,\ldots,n\}$:

$p_i \geq 0$, $\forall$ i, et $\sum_{i=1}^{n} p_i = 1$.

Supposons de plus les p_i non croissants :

$$p_i \geq p_{i+1}, \text{ pour tout } i = 1,\ldots, n. \qquad (2.15)$$

Nous avons alors les inégalités :

$$\sum_{i \succeq k} p_i \leq \sum_{i \succeq k} u_i, \tag{2.16}$$

quel que soit k entier (positif, négatif, ou nul), où le vecteur u correspond à la distribution uniforme sur N :

$$u_i = \begin{cases} 1/n, \text{ si } i \in N, \\ 0, \text{ dans le cas contraire.} \end{cases} \tag{2.17}$$

La démonstration de ce lemme trivial sera laissée au lecteur.

Théorème 2.1.

Si un modèle de rangement m est vrai pour une chaîne S, alors, pour tout T et pour tout k :

$$\alpha' \overset{\text{déf}}{=} \Pr\left[\sum_{t=1}^{T} m_t(R_t, S_{t-1}) \succ k\right] \leq U_k^{*T} \tag{2.18}$$

où les U_k^{*T} sont définis récurrentiellement par :

$$U_k^{*0} = \begin{cases} 1, \text{ si } k \prec 0, \\ 0, \text{ dans le cas contraire;} \end{cases} \tag{2.19}$$

$$U_k^{*T} = \sum_j U_j^{*T-1} u_{k-j}, \text{ pour } T \succ 0. \tag{2.20}$$

Notons que U_k^{*T} n'est autre que la probabilité pour que la somme V_T de T variables aléatoires indépendantes et uniformément distribuées sur N soit supérieure à k.

Démonstration.

Nous avons tout d'abord :

$$\alpha' = \sum_j \Pr\left[\sum_{t=1}^{T-1} m_t(R_t, S_{t-1}) = j\right] \cdot \Pr\left[m_T(R_T, S_{T-1}) \succ k-j \;\middle|\; \sum_{t=1}^{T-1} m_t(R_t, S_{t-1}) = j\right]$$

$$= \sum_j \Pr\left[\sum_{t=1}^{T-1} m_t(R_t, S_{t-1}) = j\right] \cdot \sum_{i \succ k-j} u_i ,$$

en vertu du Lemme 2.1.

En d'autres termes :

$$\alpha' \leq \Pr\Big[\sum_{t=1}^{T-1} m_t(R_t, S_{t-1}) + U \succ k\Big], \qquad (2.21)$$

où U désigne une variable aléatoire distribuée uniformément sur N, et indépendante de $\sum_{t=1}^{T-1} m_t(R_t, S_{t-1})$.

Cela étant, nous allons démontrer le théorème par induction. L'inégalité (2.18) est triviale pour T = 0. Supposons-la vraie pour T-1 ($\geq$ 0). La majoration (2.21) de α' peut encore s'écrire :

$$\sum_j \Pr\Big[\sum_{t=1}^{T-1} m_t(R_t, S_{t-1}) \succ j\Big] \cdot u_{k-j}$$

$$\leq \sum_j U_j^{*T-1} u_{k-j}, \quad \text{par induction}$$

$$= U_k^{*T}, \quad \text{en veru de (2.20).} \quad \text{C.Q.F.D.}$$

Ce théorème conduit au

Corollaire.

Si m est vrai, alors, pour tout T et pour tout $\varepsilon \succ 0$:

$$\alpha \stackrel{\text{déf}}{=} \Pr\Big[\bar{m}_T - (n+1)/2 \succ \varepsilon\Big] \leq \sigma^2 / T\varepsilon^2, \qquad (2.22)$$

où σ^2 désigne la variance d'une variable uniformément distribuée sur N :

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (i - (n+1)/2)^2. \qquad (2.23)$$

Démonstration.

$$\alpha = \Pr\Big[\sum_{t=1}^{T} m_t(R_t, S_{t-1}) \succ T(n+1)/2 + T\varepsilon\Big]$$

$$\leq \Pr\Big[V_T \succ T(n+1)/2 + T\varepsilon\Big]$$

où, en vertu du théorème 2.1, V_T représente la

somme de T variables indépendantes uniformément distribuées sur N,

$= \Pr[\, V_T - E[V_T] \succ T\varepsilon \,]$

$\leq \Pr[\, (V_T - E[V_T])^2 \succ T^2\varepsilon^2 \,]$

$\leq \sigma^2 / T\varepsilon^2$, en vertu de l'inégalité classique de Markov-Tchebitchev. C.Q.F.D.

Notons que, pour de grandes valeurs de T, une borne supérieure plus fine de α peut être obtenue grâce au théorème de tendance normale de Lévy-Lindeberg :

$$\Pr[\, V_T - E[V_T] \succ T\varepsilon \,] \simeq 1 - F_G(\varepsilon\sqrt{T}/\sigma), \qquad (2.24)$$

où F_G est la fonction de répartition gaussienne.

Etant donné une chaîne de références observée, le test présenté peut permettre une première "décantation" dans le choix d'un modèle de comportement :

1) on rejettera les modèles conduisant à un rang moyen observé supérieur à (n+1)/2;
2) entre plusieurs modèles, on préférera celui pour lequel ce rang moyen est le plus petit.

Ainsi, avec les comportements observés par Lenfant [Len74a], le modèle lru semble être un bon modèle : il conduit à un rang moyen très faible, inférieur en tout cas à n'importe quel modèle à références indépendantes. Ces observations peuvent constituer une confirmation de l'existence d'une forte corrélation entre les références consécutives. Cependant, on se gardera d'en conclure que le modèle lru est le meilleur : il peut en exister d'autres conduisant à un rang moyen moindre.

2.5. Conclusions.

Pour qui veut analyser l'allocation d'espace dans un système à mémoire virtuelle paginée, les programmes sont essentiellement des processus demandeurs de pages. Un programme a ainsi été identifié à la séquence des pages qu'il référence lors de son exécution. Une vision plus générale du comportement de programme pourrait inclure les aspects suivants :

1) Les processus de demandes de page sont considérés comme des processus à temps continu. Une page est demandée pendant une certaine durée (durée d'exécution de l'instruction) variable d'une instruction à l'autre. Un "modèle" à temps continu assez général est le modèle à réseau de retards /Len74b/. Une page est requise pendant une durée considérée comme le temps de passage du programme dans un réseau d'attente, dont chaque noeud retarde la demande d'une durée distribuée exponentiellement.
2) Chaque référence élémentaire ne porte pas nécessairement sur une et une seule page. Le code de l'instruction et les opérandes peuvent en effet se trouver sur plusieurs pages distinctes. Une extension du formalisme de ce chapitre au cas des références à pages multiples a été faite dans /TQT76b/.

En dépit de ces lacunes, le formalisme présenté est suffisant pour l'analyse d'un grand nombre de problèmes relatifs à l'allocation optimale de la mémoire, comme nous le verrons aux chapitres suivants.

En ce qui concerne la prévision des références, signalons encore les approches suivantes :

1) Avec le risque d'alourdir les procédures d'estimation, on peut envisager l'utilisation de deux modèles de prévision séparés, l'un pour la sous-chaîne des instructions, l'autre pour la sous-chaîne des données. En effet, au sein d'un programme donné, les propriétés corrélatives de ces deux sous-chaînes peuvent ne pas être suffisamment semblables pour pouvoir être modélisées par un seul et même modèle. Certains résultats expérimentaux semblent indiquer une plus forte localisation au niveau de la séquence des instructions qu'au niveau des adresses des données /LeB75/.

2) Plutôt que des modèles localisés (satisfaisant aux conditions CIL et/ou CEL), on peut rechercher des méthodes de prévision visant à la détection des boucles de programmes. Ce genre de méthodes, utilisé dans les premiers systèmes à mémoire paginée (Atlas, /KELS62/), est en général fort complexe et coûteux à implémenter. Plus récemment, les algorithmes most recently used et most frequently used, "inverses" de lru et lfu, ont été envisagés /ThI72/. Bien qu'ils puissent être justifiés pour des programmes à larges boucles, les bonnes performances généralement observées pour l'algorithme lru constituent une preuve a contrario de leur inefficacité. Nous avons proposé un algorithme adaptatif permettant de mémoriser la boucle de programme la plus récente /TQT75a/. Sa fiabilité doit encore être testée expérimentalement.

3. ALGORITHMES DE PAGINATION A PARTITION FIXE

3.1. Présentation du chapitre.

Ce chapitre étudie les principes, les propriétés, et les performances des règles décisionnelles qui gouvernent l'allocation de la mémoire centrale à des programmes, dans un contexte où chacun de ces derniers est astreint, tout au long de son exécution, à une zone fixe délimitée de la mémoire centrale. On peut certes reprocher une certaine rigidité à une telle méthode de partage de la mémoire centrale, qui limite d'une façon statique l'espace alloué à chacun des programmes. Ainsi, un programme ne peut pas céder (acquérir) de l'espace au profit (aux dépens) d'un autre. Néanmoins, c'est une méthode de sécurité parmi les plus simples garantissant à chaque programme actif une zone d'espace suffisant à son bon déroulement.

Considérons un programme s'exécutant dans un tel environnement. Initialement, toutes ses pages résident sur une mémoire secondaire, et, lors d'une première "phase", les pages sont chargées au fur et à mesure en mémoire centrale jusqu'à ce que la limite maximale d'espace alloué au programme soit atteinte. Puis, à partir de ce moment, le programme ne peut remplacer que ses propres pages. On donne le nom d'algorithme de pagination à l'ensemble des règles dynamiques qui décident, à tout instant, des pages du programme à résider en mémoire centrale, et par conséquent, des pages à charger ou à remplacer.

Au paragraphe 3.2, nous commencerons par définir le coût de la pagination d'un programme comme étant le nombre de chargements de pages encourus pendant son exécution, chaque charge-

ment provoquant une attente de transfert qui retarde le temps de réponse du programme. Il a alors été démontré que, dans la recherche des algorithmes optimaux, on peut se limiter aux algorithmes dits "à la demande" [MGST70, ADU71]. Sous ce genre d'algorithmes, une page n'est chargée que lorsqu'elle est demandée par le processeur, et un remplacement de page n'est requis que lorsque le programme occupe déjà son nombre maximum de cadres. La caractéristique d'un tel algorithme réside alors dans le choix de la page à remplacer, et le problème fondamental est de rechercher des "critères" de remplacement qui soient, sinon optimaux, du moins réalisables, simples et efficaces.

En s'inspirant notamment de l'algorithme MIN de Belady [Bel66] optimal dans le cas où les références sont connues a priori, Denning a proposé le remplacement de la page dont le délai moyen jusqu'à sa prochaine référence est le plus long [Den70, p. 178]. Un critère conceptuellement plus simple consiste à remplacer la page la moins susceptible d'être requise à la référence suivante; c'est le critère "à terme immédiat".

L'efficacité de ce critère sera justifiée, pour des programmes qui localisent leurs références dans le temps, au paragraphe 3.3. Nous y montrerons que, opérant avec une capacité de c cadres de pages, il génère un taux de défauts de page moindre que tout algorithme non prospectif travaillant avec c-1 pages.

Enfin, aux paragraphes 3.4 et 3.5, nous analyserons les performances de ce critère à travers deux modèles de comportement assez généraux inspirés de la prévision économétrique :

le modèle de lissage exponentiel, et le modèle de lissage sur les rangs. Pour ces modèles, qui généralisent ceux, classiques, à références indépendantes et à liste lru, le critère à terme immédiat est d'ailleurs équivalent au critère du plus long délai moyen de récurrence. Pour le modèle de lissage sur les rangs, nous dériverons le taux de défauts de page pour diverses valeurs de la mémoire allouée. Pour le modèle de lissage exponentiel, nous établirons l'optimalité de ce critère, après en avoir donné une approximation par excès du taux de défauts de page.

3.2. Formulation des algorithmes de pagination.

3.2.1. Le problème du remplacement.

De façon informelle, un algorithme de pagination, opérant avec une capacité-mémoire de c cadres de pages et sur une chaîne de références $S = R_1 \ldots R_t \ldots$, est une certaine règle de décision AP qui génère une séquence d'états-mémoire :

$$\{E_t(c, AP);\ t=0,1,2,\ldots\} \tag{3.1}$$

Chaque $E_t(c, AP)$, qui représente l'ensemble des pages en mémoire centrale lors de l'exécution de la $t^{ième}$ opération élémentaire, est une partie de X, l'ensemble des pages du programme. Cette séquence doit évidemment satisfaire aux conditions suivantes :

$$|E_t(c, AP)| \leq c, \ \forall\ t \geq 0, \text{ et} \tag{3.2}$$

$$E_t(c, AP) \ni R_t, \ \forall\ t > 0. \tag{3.3}$$

Un AP est dit réalisable (APR), ou non prospectif, si, pour tout t, $E_t(c, AP)$ ne dépend que de $S_t = R_1 \ldots R_t$ (et de l'état initial $E_0(c, AP)$). Pour une définition plus formelle des algorithmes de pagination à partition fixe, on pourra se référer à [ADU71].

Dans ce paragraphe, le coût d'un AP sera pris comme le nombre de chargements de pages encourus pendant une certaine période de temps (par exemple, la durée d'exécution du programme). Raisonnablement, on peut s'attendre à ce que les performances globales du système (taux d'utilisation du processeur, temps de réponse moyen des programmes,...) sont d'autant meilleures que chacun des programmes est rarement interrompu par des attentes de transfert de pages.

Le coût d'un AP, encouru pendant une période $[t, T]$, l'état de la mémoire à l'instant t-1 étant $E_{t-1}(c, AP) = e$, sera noté :

$$F_{[t, T]}(e, c, AP) = \sum_{u=t}^{T} |E_u(c, AP) - E_{u-1}(c, AP)|. \quad (3.4)$$

Cette fonction de coût permet une grande simplification dans la recherche des algorithmes optimaux : on peut limiter ses investigations à une classe d'algorithmes de pagination dits "à la demande" (demand paging).

Définition 3.1.

Un algorithme de pagination est dit à la demande (APD) si, pour tout t, $e' = E_t(c, APD)$ est relié à $e = E_{t-1}(c, APD)$ par une relation de la forme :

$$e' = \begin{cases} e & \text{si } R_t \in e, \\ e + R_t & \text{si } R_t \notin e \text{ et } |e| < c, \\ e + R_t - z_t(c), & \text{si } R_t \notin e \text{ et } |e| = c, \end{cases}$$

$z_t(c)$ désignant une certaine page dans e (+ est mis pour l'union d'ensembles disjoints, et les crochets sont omis pour les ensembles à un seul élément).

Théorème 3.1. (/MGST70, ADU71/)

Pour tout algorithme de pagination AP, il existe un algorithme à la demande APD tel que

$$F_{[t, T]}(e, c, APD) \leq F_{[t, T]}(e, c, AP),$$

quels que soient S, e, c, t, et T.

De plus, si AP est un algorithme réalisable, il en sera de même pour APD.

Notons que, sous une pagination à la demande, le nombre de chargements de page s'identifie au nombre de défauts de page ($|E_t(c, APD) - E_{t-1}(c, APD)|$ ne peut valoir que 0 ou 1, selon que R_t appartient à $E_{t-1}(c, APD)$ ou non).

Il est évident que la définition 3.1 ne suffit pas pour caractériser complètement un APD : il reste à spécifier l'identité de la page à remplacer $z_t(c)$. Pour un grand nombre d'algorithmes, ce choix se fait par l'intermédiaire d'un certain "ordre de préférence" sur X : la page à remplacer est alors la "dernière" dans cet ordre. Ce genre d'algorithmes sera présenté au § 3.2.2. Un problème fondamental de ce chapitre est de déterminer des algorithmes de remplacement sinon optimaux, du moins assez efficaces. Ce problème sera discuté, en relation avec le comportement de programme, au § 3.2.3.

3.2.2. Algorithmes déduits d'un modèle de rangement.

Tout d'abord, un ordre de préférence sur X peut être spécifié par un modèle de rangement. Avec les notations de la définition 3.1, nous adopterons la terminologie suivante.

Définition 3.2.

Etant donné un modèle de rangement m, nous désignerons par M l'APD pour lequel la page à remplacer $z = z_t(c)$ est celle dans e de rang le plus élevé :

$$m_{t+1}(z, S_t) = \max_{y \in e} m_{t+1}(y, S_t).$$

Des exemples triviaux de ces algorithmes non prospectifs peuvent être obtenus à partir des modèles du § 2.3.3. Ainsi

— Pour le modèle à références indépendantes, le modèle de rangement induit m est stationnaire : $m_t(., s_{t-1})$ ne dépend pas de (t, s_{t-1}). M est l'algorithme A_0 d'Aho et al /ADU71/. Sous cet algorithme, la page à remplacer est la dernière d'une liste de priorité statique.

— Le modèle lru conduit à l'algorithme LRU. La page à remplacer est celle utilisée la moins récemment.

— Si m est induit par un modèle lfu, alors M remplace la page utilisée la moins fréquemment; c'est l'algorithme LFU (least frequently used).

Ces algorithmes de pagination à la demande, déduits à partir d'un modèle de rangement, et encore appelés algorithmes à liste de priorité, ont fait l'objet d'une étude remarquable de Mattson et al /MGST70/. Le théorème suivant y a été établi :

Théorème 3.2.

Tout algorithme M déduit d'un modèle de rangement m est un algorithme à pile (stack algorithm), c'est-à-dire possède la propriété d'inclusion :

$$E_t(c-1, M) \subset E_t(c, M), \quad \forall\, c \geq 1,\ t \geq 0. \tag{3.5}$$

Désignons par

$$y_t(c) = E_t(c, M) - E_t(c-1, M). \tag{3.6}$$

La page $z_t(c)$ à remplacer de $E_{t-1}(c, M)$ (cfr déf. 3.1) satisfait :

$$m_{t+1}(z_t(c), S_t) = \max \{m_{t+1}(z_t(c-1), S_t), m_{t+1}(y_t(c), S_t)\}. \tag{3.7}$$

Nous terminerons ce paragraphe en signalant qu'il existe des APD qui, bien que reposant sur un "critère de préférence", ne satisfont pas à la définition 3.2, et par conséquent pas au théorème 3.2 non plus. Il en est ainsi notamment de l'algorithme FIFO (first in first out) : la page à remplacer est celle ayant séjourné depuis le plus longtemps en mémoire centrale. On peut en effet exhiber des chaînes de références particulières pour lesquelles cet ordre FIFO ne dépend pas uniquement des références passées comme dans le cas d'un modèle de rangement, mais aussi du nombre c de cadres alloués. D'ailleurs, pour ces chaînes, on peut constater l'anomalie suivante : sous l'algorithme FIFO, le taux de défauts de page ne décroît pas nécessairement avec c /BNS69/. Notons que cette anomalie est impossible sous les algorithmes à liste de priorité, en vertu de l'inclusion (3.5).

D'autres exemples d'algorithmes qui ne sont pas à liste de priorité peuvent être fournis par les algorithmes pour lesquels la page à remplacer est choisie d'une manière non déterministe (c'est-à-dire selon une certaine distribution de probabilité) parmi les pages en mémoire. Il en est notamment ainsi de l'algorithme RAND de Belady /Bel66/, et plus généralement des algorithmes de remplacement aléatoires à préchargement partiel introduits par Gelenbe /Gel73a/.

3.2.3. Critères de remplacement.

Un problème clé dans la conception des algorithmes de pagination (à la demande) est de choisir un critère spécifiant la page à remplacer, et cela en vue de minimiser le coût (3.4).

Ce problème a d'abord été étudié dans le cas déterministe où le processus des références est connu avec certitude à l'avance. Belady [Bel66] avait suggéré le remplacement de la page qui sera référencée en dernier lieu. C'est l'algorithme MIN, ou OPT, dont l'optimalité a été établie par Mattson et al [MGST70] et par Aho et al [ADU71].

Dans le contexte stochastique, le problème de la recherche des algorithmes réalisables optimaux peut être formulé comme un problème de programmation dynamique [ADU71]. Pour un processus des références obéissant à un modèle explicite p, les coûts moyens optimaux

$$\tilde{F}_t(e, s_{t-1}) = \min_{APR} E\left[F_{[t, T]}(e, c, APR) \mid S_{t-1}=s_{t-1}\right] \quad (3.8)$$

(c et T étant fixés) doivent satifaire, en vertu du principe d'optimalité et de l'optimalité des AP à la demande (théorème 3.1), aux équations fonctionnelles suivantes, lorsque $|e| = c$:

$$\begin{aligned} \tilde{F}_t(e, s_{t-1}) &= \sum_{r \in e} p_t(r, s_{t-1}) \tilde{F}_{t+1}(e, s_{t-1}.r) \\ &+ \sum_{r \notin e} p_t(r, s_{t-1}) \{1 + \min_{y \in e} \tilde{F}_{t+1}(e+r-y, s_{t-1}.r)\}, \end{aligned} \quad (3.9)$$

pour $t < T$, et :

$$\tilde{F}_T(e, s_{T-1}) = \sum_{r \notin e} p_T(r, s_{T-1}). \quad (3.10)$$

Ce problème est en général assez complexe, et n'a été résolu que pour certaines structures probabilistes particulières. Ainsi, pour le modèle à références indépendantes, l'algorithme optimal est l'algorithme A_0 sous lequel la page à remplacer est est celle ayant la plus petite probabilité de référence /ADU71/, tandis que pour le modèle à liste lru, l'algorithme optimal est l'algorithme LRU /CoD73, chap. 6/.

En se contentant de règles raisonnables plutôt qu'optimales, Denning /Den70, p. 178/ suggère la transposition suivante du critère déterministe MIN : la page à remplacer est choisie comme celle ayant le plus long délai moyen de récurrence. Il est à noter que ce critère conduit à l'algorithme A_0 sous un modèle à références indépendantes, et à l'algorithme LRU sous un modèle à liste lru. Néanmoins, il est assez complexe du point de vue opérationnel. La donnée d'un modèle de comportement de programme sous une forme explicite est nécessaire; les délais moyens de récurrence des pages sont alors estimés par la procédure décrite par les équations (2.1-4).

Nous avons proposé une alternative bien plus simple consistant à remplacer la page la moins susceptible d'être requise à la référence suivante /TQT76c/. A première vue, ce critère présente, comparativement au critère de la page à plus long délai moyen de récurrence ou à tout autre critère "raisonnable", les caractéristiques suivantes :

- à chaque instant, il vise une (sous-)optimisation à très court terme, voire même à une seule étape, alors que le problème d'optimisation est à horizon déterminé;

- à partir d'un modèle explicite du comportement, ce critère à terme immédiat est direct, tandis que les autres critères nécessitent généralement des calculs auxiliaires plus ou moins complexes; dans ce sens, c'est peut être le critère le plus simple qui soit;
- un modèle quantifiant explicitement les chances de référence de chacune des pages n'est même pas nécessaire; il suffit d'un modèle de rangement, tel que ceux définis au § 2.2.2, permettant de générer un ordre dynamique sur l'ensemble des pages.

Si la simplicité du critère à terme immédiat peut déjà pallier à son inefficacité éventuelle, il reste à déterminer des conditions pratiques sous lesquelles on peut s'attendre à de bonnes performances de la part de ce critère. Dans le restant de ce chapitre, nous tenterons principalement de justifier l'efficacité de ce critère, pour des comportements de programme satisfaisant aux conditions de localisation du § 2.3.

Signalons que, avec la définition 3.2, si un modèle de rangement m est vrai, alors M remplace la page la moins susceptible d'être requise à la prochaine étape, conformément au critère à terme immédiat. Pour la commodité, nous adopterons la

Définition 3.2-bis.

> Si m est induit par un modèle explicite p, nous dirons que M est un APD induit par p.

Dans ce cas, M remplace la page possédant la plus petite prévision de référence à la prochaine étape, prévision générée par p.

3.3. Pagination sous des modèles de rangement localisés.

Nous analysons dans ce paragraphe les performances de l'algorithme M dans le cas où m satisfait à la CIL. Le théorème 3.3 présente une justification du critère à terme immédiat : si m est vrai, alors M est sous-optimal en ce sens que, opérant avec une capacité de c cadres, il est meilleur que n'importe quel algorithme réalisable qui ne dispose que de c-1 cadres. En effet, dans ce cas, à partir d'un certain instant, les c-1 pages de rangs les moins élevés, donc les plus susceptibles d'être prochainement référencées, sont toutes en mémoire centrale.

Définition 3.3.

Etant donné un modèle de rangement m pour une chaîne S, les localités instantanées sont constituées, à chaque instant, des pages de rangs les moins élevés :

$$L_t(c') = \{x \mid x \in X \text{ et } m_{t+1}(x, S_t) \leq c'\}, \qquad (3.11)$$

$$t = 0, 1, \ldots, \text{ et } c' = 0, 1, \ldots$$

Avant de démontrer le théorème 3.3, examinons comment ces localités varient sous la CIL :

$$L_{t+1}(c') - L_t(c') = \{x \mid m_{t+2}(x, S_{t+1}) \leq c' < m_{t+1}(x, S_t)\}$$

$$\subset \{R_{t+1}\}, \qquad (3.12)$$

en vertu de la CIL. Ainsi, $L_{t+1}(c')$ ne peut différer de $L_t(c')$ que de la page référencée R_{t+1} tout au plus. De plus, si $R_{t+1} \in L_t(c')$, alors l'inclusion précédente implique :

$$L_{t+1}(c') = L_t(c'),$$

puisque ces deux ensembles ont le même cardinal c'.

D'une façon imagée, nous pouvons donc dire que la CIL suppose que les variations des localités d'une étape à l'autre sont assez progressives. Des renouvellements brusques des localités "réelles" (pages momentanément privilégiées) ne peuvent être détectés qu'après plusieurs références. Notons encore que cette dynamique des $L_t(c')$ est analogue à celle des états de la mémoire générés sous des algorithmes à la demande (définition 3.1).

Après cette définition préliminaire, la démonstration de la sous-optimalité de M repose essentiellement sur le lemme suivant.

Lemme 3.1.

Soient S une chaîne de références et m un modèle implicite satisfaisant à la CIL. L'algorithme M jouit alors de la propriété suivante : l'inclusion

$$U_t \cap L_t(c-1) \subset E_t(c, M) \qquad (3.13)$$

est vérifiée pour tout $t = 0, 1, \ldots$ et pour tout $c = 1, 2, \ldots$;
U_t représente l'ensemble des pages ayant été référencées au moins une fois jusqu'à l'instant t :

$$U_t = \bigcup_{v=1}^{t} R_v .$$

Démonstration.

Pour $t = 0$, l'inclusion à démontrer est triviale pour tout c puisque $U_0 = \emptyset$. Supposons-la vraie à t-1, c'est-à-dire que

$$U_{t-1} \cap L_{t-1}(c-1) \subset E_{t-1}(c, M), \quad \forall\, c = 1, 2, \ldots$$

En utilisant cette hypothèse d'induction, il faut montrer que, si

$$x \in U_t \quad \text{et} \quad x \in L_t(c-1), \qquad (3.14)$$

alors $x \in E_t(c, M)$, pour tout x et pour tout c.

Pour établir cette implication, plusieurs cas sont à distinguer.

1. $x \in E_{t-1}(c, M)$.

   1.1. Si aucun remplacement n'a lieu à l'instant t, alors, en invoquant le fait que M est "à la demande", nous avons $x \in E_t(c, M)$ aussi.

   1.2. Si un remplacement a lieu, la page remplacée z est de rang maximal dans $E_{t-1}(c, M)$, et, puisque M est à la demande, $E_{t-1}(c, M)$ contient c pages. Par conséquent $m_{t+1}(z, S_t) \geq c$.

   x n'est donc pas z, puisque, en vertu de (3.14), $m_{t+1}(x, S_t) \leq c-1$.

   x reste par conséquent en mémoire à l'instant t : $x \in E_t(c, M)$.

2. $x \notin E_{t-1}(c, M)$.

   L'hypothèse d'induction implique alors que $x \notin U_{t-1}$ ou $x \notin L_{t-1}(c-1)$.

   2.1. Si $x \notin U_{t-1}$, alors $x = R_t$ (puisque, en vertu de (3.14) : $x \in U_t$), et x doit être chargée dans $E_t(c, M)$.

   2.2. Si $x \notin L_{t-1}(c-1)$, alors $m_t(x, S_{t-1}) > c-1$.

   Or, en vertu de (3.14) : $m_{t+1}(x, S_t) \leq c-1$.

   Donc : $m_{t+1}(x, S_t) < m_t(x, S_{t-1})$.

   Puisque m satisfait à la CIL, x ne peut donc être que la page référencée R_t, et $x \in E_t(c, M)$. C.Q.F.D.

Notons que l'inclusion déterministe (3.13) exige seulement que m satisfasse à la CIL. Si, de plus, m est vrai pour S, alors nous avons le

Théorème 3.3.

Si un modèle m, satisfaisant à la CIL, est vrai pour une chaîne S, alors, pour tout algorithme réalisable APR, et pour tout c = 2, 3,..., nous avons :

$$\Pr[R_{t+1} \notin E_t(c, M)] \leq \Pr[R_{t+1} \notin E_t(c-1, APR)], \quad (3.15)$$

pour tout $t \geq v$, v étant un instant tel que

$$L_v(c-1) \subset U_v. \quad (3.16)$$

Démonstration.

L'inégalité (3.15) est immédiate si, à l'instant t, nous avons :

$$L_t(c-1) \subset U_t. \quad (3.17)$$

En effet, dans ce cas, l'inclusion du lemme 3.1 se réduit à

$$L_t(c-1) \subset E_t(c, M), \quad (3.18)$$

de sorte que

$$\Pr[R_{t+1} \notin E_t(c, M)] \leq \Pr[R_{t+1} \notin L_t(c-1)]. \quad (3.19)$$

Mais, m étant vrai pour S, $L_t(c-1)$, fonction de S_t seulement, est constitué précisément des c-1 pages de probabilités de référence (à l'instant t+1) les plus élevées. Par conséquent, puisque $E_t(c-1, APR)$ ne peut contenir que c-1 pages tout au plus :

$$\Pr[R_{t+1} \notin L_t(c-1)] \leq \Pr[R_{t+1} \notin E_t(c-1, APR)]. \quad (3.20)$$

L'inégalité (3.15) résulte alors de (3.19) et (3.20).

Il reste seulement à établir l'inclusion (3.17) pour tout $t \geq v$. Par hypothèse, elle est vraie à l'instant v. Supposons-la vraie à $t-1 \geq v$. Alors :

$$\begin{aligned} L_t(c-1) &= (L_t(c-1) - L_{t-1}(c-1)) \cup (L_t(c-1) \cap L_{t-1}(c-1)) \\ &\subset R_t \cup L_{t-1}(c-1), \text{ en vertu de (3.12)} \\ &\subset R_t \cup U_{t-1}, \text{ par induction} \\ &= U_t. \quad \text{C.Q.F.D.} \end{aligned}$$

3.4. Pagination sous le modèle à rangs indépendants.

Nous considérons dans cette section un modèle de comportement qui généralise à la fois le modèle à références indépendantes et le modèle à liste lru. Ce modèle sera appelé modèle à rangs indépendants car, pour des chaînes obéissant à ce modèle, les rangs des pages consécutivement référencées sont indépendants entre eux et identiquement distribués. Nous montrerons d'abord que, pour ce genre de modèles, le critère à terme immédiat est équivalent à tout autre critère "raisonnable" : les pages de rangs les plus élevés sont aussi celles les moins susceptibles d'être requises jusqu'à tout horizon h, et possèdent les plus longs délais moyens de récurrence. Nous donnerons également le taux de défauts de page sous l'algorithme induit. En particularisant ce résultat au modèle à références indépendantes ou à celui à liste lru, on retrouve les expressions bien connues du taux de défauts de page sous les algorithmes A_0 ou LRU.

3.4.1. Le modèle.

Considérons d'abord un modèle de rangement satisfaisant aux conditions suivantes :

C1. A tout instant, le rangement actuel et le rang de la page référencée suffisent à déterminer le rangement à l'instant suivant. En d'autres termes, pour tout $(t, s_{t-1}) \in Q$ et $r \in X$ soit :

$k = m_t(r, s_{t-1})$;

le rangement $m_{t+1}(., s_{t-1}.r)$ se déduit de $m_t(., s_{t-1})$ par une permutation β_k ne dépendant que de k :

$m_{t+1}(x, s_{t-1}.r) = \beta_k(m_t(x, s_{t-1}))$, $\forall\, x \in X$.

C2. Le rang de la page référencée ne croît jamais :

$\beta_k(k) \leq k$, $\forall\, k = 1,\ldots,n$;

C3. Les autres pages conservent leur rang relatif :

pour i, j, $k \in \{1,\ldots,n\}$,

si $i \neq k$, $j \neq k$ et $i < j$, alors $\beta_k(i) < \beta_k(j)$.

Le modèle à rangs indépendants est alors défini en ajoutant à ces conditions la condition suivante :

C4. Les prévisions d'occurrence des pages ne dépendent que de leur rang actuel. En d'autres termes, pour tous $(t, s_{t-1}) \in Q$ et $x \in X$:

$p_t(x, s_{t-1}) = a_{m_t(x, s_{t-1})} > 0$,

où $a = \{a_1,\ldots, a_n\}$ est un vecteur de probabilité fixé.

La localisation des références peut être tenue compte en s'imposant, en plus, la condition

C5. $a_i \geq a_{i+1}$, pour tout $i = 1,\ldots, n-1$.

Trivialement, les conditions C2 et C3 impliquent la CIL. Il est également évident que les conditions C1-C4 définissent un modèle explicite qui se réduit, par exemple, à

- un modèle à références indépendantes si $\beta_k(k) = k$, $\forall\, k$,
- un modèle à liste lru si $\beta_k(k) = 1$, $\forall\, k$.

En faisant varier les valeurs des $\beta_k(k)$ dans $[1, k]$, on peut obtenir toute une gamme de modèles reflétant d'autant mieux la localisation que les $\beta_k(k)$ sont plus petits.

Notons également que, si de plus la condition C5 est satisfaite, alors le modèle à rangs indépendants p induit précisément le modèle m.

3.4.2. Equivalence des critères de remplacement.

Avant d'analyser les performances de l'algorithme induit, montrons d'abord que, pour des modèles satisfaisant aux conditions C1-C5, la page à remplacer sous l'algorithme induit est aussi celle possédant le plus long délai moyen de récurrence. La démonstration utilise le

Lemme 3.2.

Tout modèle satisfaisant à C2 et C3 satisfait aussi à :

C2' : $i \prec j \Rightarrow \beta_j(i) \leq \beta_i(j)$, $\forall$ i, j = 1,..., n.

Ce lemme sera démontré dans l'annexe à ce chapitre.

Théorème 3.4.

Pour un modèle satisfaisant aux conditions C1, C2', C3-C5, alors, quels que soient $(t, s_{t-1}) \in Q$, x, y $\in X$, et h=1,2,..., nous avons l'implication :

$$m_t(x, s_{t-1}) \prec m_t(y, s_{t-1}) \Rightarrow q_t^{(h)}(x, s_{t-1}) \leq q_t^{(h)}(y, s_{t-1})$$

$$(\Rightarrow \bar{p}_t(x, s_{t-1}) \leq \bar{p}_t(y, s_{t-1}), \text{ en vertu de (2.4)}).$$

Démonstration.

Pour un tel modèle, il est clair que les prévisions $q_t^{(h)}(x, s_{t-1})$, définies par les équations (2.1) et (2.2), ne dépendent que du rang $m_t(x, s_{t-1})$ et de h. Pour tous (t, s_{t-1}), x et h, posons :

$$q^{(h)}(m_t(x, s_{t-1})) = q_t^{(h)}(x, s_{t-1}).$$

Il faut et il suffit de démontrer que, pour tout h :

$$i \prec j \Rightarrow q^{(h)}(i) \leq q^{(h)}(j).$$

Cette dernière propriété est triviale pour h = 1, puisque $q^{(1)}(i) = 1 - a_i$. En la supposant vraie pour h-1 (hypothèse d'induction), nous devons l'établir pour h. En vertu de (2.2) :

$$q^{(h)}(i) - q^{(h)}(j) = \sum_{k \neq i,j} a_k \{ q^{(h-1)}(\beta_k(i)) - q^{(h-1)}(\beta_k(j)) \}$$
$$+ a_j . q^{(h-1)}(\beta_j(i)) - a_i . q^{(h-1)}(\beta_i(j)).$$

En vertu de C3, puisque $i \prec j$, nous avons également $\beta_k(i) \prec \beta_k(j)$ pour tout $k \neq i, j$. Les termes entre accollades sont alors négatifs ou nuls, par induction.

Les deux derniers termes conduisent également à une valeur non positive :

- d'une part, en vertu de C5 : $a_j \prec a_i$;
- d'autre part, en vertu de C2' : $\beta_j(i) \preceq \beta_i(j)$, d'où :

$q^{(h-1)}(\beta_j(i)) \preceq q^{(h-1)}(\beta_i(j))$, par induction. C.Q.F.D.

Remarquons que ce théorème, qui requiert C2' plutôt que C2, peut s'appliquer à des modèles pour lesquels le rang de la page référencée peut croître. Il en est ainsi, notamment, du modèle "à liste mru" qui satisfait à C1, C3-C5, et tel que

$\beta_k(k) = n$, pour tout $k = 1, \ldots, n$.

(la condition C2' est en effet satisfaite : pour $i \prec j$, $\beta_j(i) = i \preceq \beta_i(j) = j-1$).

3.4.3. Taux de défauts de page.

Considérons maintenant une chaîne de références $S = R_1 \ldots R_t \ldots$ obéissant à un modèle explicite qui satisfait aux conditions C1-C4. Désignons par M l'algorithme qui remplace la page de rang le plus élevé (les rangements satisfont au conditions C1-C3 Nous allons étudier les taux de défauts de page sous M :

$$f(c) = \lim_{t \to \infty} \Pr[R_t \notin E_{t-1}(c, M)], \quad c = 1, 2, \ldots, n. \qquad (3.21$$

Théorème 3.5.

Pour une chaîne de références S satisfaisant aux conditions C1, C2, C3, et C4, les taux de défauts de page générés par l'algorithme M au cours de la pagination de cette chaîne valent :

$$f(c) = \sum_{i=c}^{n} a_i.(1 - g(i, c)), \quad c = 1, 2, \ldots, n, \tag{3.22}$$

où les g sont donnés par :

$$g(c, c) = \sum_{k \to \beta_k(k) \leq c \leq k} a_k \Big/ \sum_{k=c}^{n} a_k, \tag{3.23}$$

et, pour $i \succ c$:

$$g(i, c) = \sum_{k \to \beta_k(k) = i} a_k \Big/ \sum_{k=c}^{n} a_k. \tag{3.24}$$

Démonstration.

Pour tous t et c, nous avons :

$$\begin{aligned} \Pr[R_t \notin E_{t-1}(c, M)] &= \sum_{i=1}^{n} \Pr[R_t = m_t^{-1}(i, S_{t-1})] \\ &\qquad \times \Pr[m_t^{-1}(i, S_{t-1}) \notin E_{t-1}(c,M)] \\ &= \sum_{i=1}^{n} a_i.(1 - g_t(i, c)), \end{aligned} \tag{3.25}$$

où $g_t(i, c)$ désigne la probabilité pour que la page de rang i à l'instant t soit dans $E_{t-1}(c, M)$:

$$g_t(i, c) = \Pr[m_t^{-1}(i, S_{t-1}) \in E_{t-1}(c, M)].$$

Tout d'abord, puisque tous les a_i sont strictement positifs, $\lim_{t \to \infty} \Pr[U_t = X] = 1$. Comme C2 et C3 impliquent la CIL, nous avons en vertu du Lemme 3.1 : $\lim \Pr[L_t(c-1) \subset E_t(c, M)] = 1$. Autrement dit, en régime asymptotique, toutes les pages de rang inférieur à c sont en mémoire avec la probabilité 1 :

$$\lim_{t \to \infty} g_t(i, c) = 1, \text{ pour } i \prec c. \tag{3.26}$$

(3.25) et (3.26) justifient la forme (3.22) de f(c).

Soit maintenant un instant t tel que :

$$L_{t-1}(c-1) \subset E_{t-1}(c, M) \tag{3.27}$$

(un tel instant existe et est fini avec la probabilité 1).

Désignons par k le rang de la page référencée à l'instant t :

$$k = m_t(R_t, S_{t-1}). \tag{3.28}$$

Le lemme 3.3 qui va suivre établira les équivalences suivantes :

$$m_{t+1}^{-1}(c, S_t) \in E_t(c, M)$$

$$\Leftrightarrow \text{ ou } \begin{cases} m_t^{-1}(c, S_{t-1}) \in E_{t-1}(c, M) \text{ et } k \prec c, \\ \beta_k(k) \preccurlyeq c \preccurlyeq k; \end{cases} \tag{3.29}$$

et, pour $i \succ c$:

$$m_{t+1}^{-1}(i, S_t) \in E_t(c, M)$$

$$\Leftrightarrow \text{ ou } \begin{cases} m_t^{-1}(i, S_{t-1}) \in E_{t-1}(c, M) \text{ et } k \prec c, \\ \beta_k(k) = i \text{ (et } k \succcurlyeq c). \end{cases} \tag{3.30}$$

De (3.29), nous tirons :

$$g_{t+1}(c, c) = g_t(c, c) \cdot \sum_{k \prec c} a(k) + \sum_{k \succ \beta_k(k) \preccurlyeq c \preccurlyeq k} a(k),$$

et, de (3.30), pour $i \succ c$:

$$g_{t+1}(i, c) = g_t(i, c) \cdot \sum_{k \prec c} a(k) + \sum_{k \succ \beta_k(k) = i} a(k).$$

De ces deux dernières formules, il résulte que, pour tout $i \succcurlyeq c$, les limites

$$g(i, c) = \lim_{t \to \infty} g_t(i, c)$$

existent, et sont données par les expressions (3.23) et (3.24).

C.Q.F.D.

Lemme 3.3.

Sous C1-C3, et si t satisfait à (3.27), alors nous avons les équivalences (3.29) et (3.30), où k est donné par (3.28).

La démonstration de ce lemme sera donnée en annexe.

Notons les particularisations suivantes du théorème 3.5.

- Pour un modèle à références indépendantes, $\beta_k(k) = k$ pour tout k. Nous avons alors :

  $$g(i, c) = a_i / \sum_{k=c}^{n} a_k, \quad \text{pour tout } i = c, c+1, \ldots, n,$$

  et les taux de défauts de page sous M, qui n'est autre que l'algorithme A_0, valent

  $$f(c) = \sum_{i=c}^{n} a_i . (1 - a_i / \sum_{k=c}^{n} a_k). \qquad (3.31)$$

- Si $\beta_k(k) \leq c$, pour tout k, alors :

  $g(c, c) = 1$, et $g(i, c) = 0$ pour tout $i < c$.

  Les taux de défauts de page sous M sont donnés par :

  $$f(c) = \sum_{i=c+1}^{n} a_i . \qquad (3.32)$$

  Cette expression est aussi celui du taux de défaut de page sous l'algorithme LRU pour le modèle à liste lru [CoD73].

Notons également que le théorème 3.5 ne requiert pas la condition C5 selon laquelle les a_i sont non croissants. Si C5 est vraie, le modèle de rangement induit l'est également. M est alors sous optimal, en vertu du théorème 3.3. En fait, l'optimalité de M a été établie pour les modèles à références indépendantes et à liste lru [ADU71, CoD73].

3.5. Pagination sous le modèle de lissage exponentiel.

Nous présentons dans cette section un modèle explicite de prévision du comportement de programme, inspiré des méthodes de lissage utilisées dans la prévision des séries chronologiques, le modèle de lissage exponentiel. Ce modèle contient, comme cas particuliers, certains modèles classiques comme le modèle à références indépendantes, le modèle lfu, et le modèle lru. Un premier résultat démontre que, pour ce modèle, les pages possédant les plus petites prévisions de référence dans l'immédiat sont aussi celles dont les délais moyens de récurrence sont les plus longs. Ensuite, nous analyserons les performances de l'algorithme induit par ce modèle. Nous établirons alors une approximation par excès du taux de défauts de page qui permettra une discussion du taux de défauts de page en fonction du nombre de cadres alloués et de la taille des pages. Nous démontrerons enfin l'optimalité de l'algorithme induit.

3.5.1. Le modèle.

Soient $\{p_1(x);\ x \in X\}$ un vecteur de prévision initial, et $\alpha_1.\alpha_2 \ldots \alpha_t \ldots$ une suite de coefficients réels tels que $\alpha_t \in [0, 1]$ pour tout $t = 1, 2, \ldots$ Les prévisions consécutives sont alors données par l'expression récurrentielle :

$$p_{t+1}(x, s_{t-1}.r) = (1-\alpha_t)p_t(x, s_{t-1}) + \begin{cases} \alpha_t, \text{si } x = r \\ 0, \text{ si } x \neq r \end{cases}, \quad (3.33)$$

quels que soient x, r, t, et s_{t-1}.

Ce modèle satisfait trivialement à CEL, et induit des modèles de rangement satisfaisant à la CIL.

Les coefficients de lissage α_t peuvent être interprétés comme des mesures instantanées de la tendance à la localisation. Plus on veut accorder de "poids" aux références récentes dans la méthode de prévision, plus on choisira ces coefficients élevés. Notamment :

- Si $\alpha_t = 0$ pour tout t, alors les prévisions instantanées de chaque page restent invariantes dans le temps, et le modèle est tout simplement un modèle à références indépendantes.

- Si $\alpha_t = 1/t$, alors (3.33) se réduit à (2.13), et le modèle, à un modèle lru.

- Si $\alpha_t \geq 1/2$ pour tout t, alors on peut facilement démontrer l'implication (2.8), et le modèle induit est un modèle lru.

Dans ce dernier cas, une autre propriété du modèle, en relation avec l'ensemble d'activité (working set), sera examinée au chapitre suivant. Nous nous limiterons ici à l'étude des propriétés et des performances de l'algorithme induit par ce genre de modèles.

3.5.2. Equivalence des critères.

Pour le modèle de lissage défini par (3.33), montrons également que les pages les moins susceptibles d'être référencées à la prochaine étape le sont aussi jusqu'à tout horizon h. Il en résulte notamment que la page à remplacer sous l'algorithme induit est aussi celle à plus long délai moyen de récurrence.

Théorème 3.6.

Pour un modèle de lissage défini par (3.33), alors, quels que soient $(t, s_{t-1}) \in Q$, $x, y \in \mathcal{X}$, et $h = 1, 2, \ldots$, nous avons l'implication :

$$p_t(x, s_{t-1}) \succ p_t(y, s_{t-1}) \Rightarrow q_t^{(h)}(x, s_{t-1}) \preccurlyeq q_t^{(h)}(y, s_{t-1}).$$

$$(\Rightarrow \bar{p}_t(x, s_{t-1}) \leq \bar{p}_t(y, s_{t-1}), \text{ en vertu de (2.4)}).$$

Démonstration.

Pour ce modèle, il est clair que les prévisions $q_t^{(h)}(x, s_{t-1})$ ne dépendent que de $p_t(x, s_{t-1})$ et de h. Pour tous (t, s_{t-1}), x, et h, posons :

$$q_t^{(h)}(p_t(x, s_{t-1})) = q_t^{(h)}(x, s_{t-1}).$$

Il faut et il suffit de démontrer que, quels que soient h et t, $q_t^{(h)}(\pi)$ est une fonction non croissante avec π, c'est-à-dire que :

$$\frac{d}{d\pi} q_t^{(h)}(\pi) \leq 0, \quad \forall \pi, t, \text{ et } h.$$

Cette propriété est triviale pour h = 1 (en effet $q_t^{(1)}(\pi) = 1-\pi$). Supposons-la vraie pour h-1, quels que soient π et t (hypothèse d'induction).

En dérivant par rapport à π l'expression

$$q_t^{(h)}(\pi) = (1-\pi) q_{t+1}^{(h-1)}((1-\alpha_t)\pi),$$

nous obtenons, après avoir posé $\pi' = (1-\alpha_t)\pi$:

$$\frac{d}{d\pi} q_t^{(h)}(\pi) = - q_{t+1}^{(h-1)}(\pi') + (1-\pi)(1-\alpha_t) \frac{d}{d\pi'} q_{t+1}^{(h-1)}(\pi').$$

Dans cette dernière expression, $q_{t+1}^{(h-1)}(\pi')$, qui représente une probabilité, est positif ou nul. Quant au second terme, il est négatif ou nul, par induction. C.Q.F.D.

3.5.3. Taux de défauts de page.

Considérons maintenant une chaîne de références S obéissant à un modèle de lissage. Les deux propriétés suivantes seront utiles pour une discussion du taux de défauts de page sous l'algorithme induit M, en fonction de l'espace alloué au programme et de la taille des pages.

Propriété 1.

Considérons une partition $X' = \{x'_1, \ldots, x'_n,\}$ de X (les x'_j sont des sous-ensembles disjoints de X, et leur union donne X). Au niveau de cette partition X', considérons la chaîne des références : $S' = R'_1 \ldots R'_t \ldots$, où, pour tout t, $R'_t = x'$ si et seulement si $R_t \in x'$.

Les prévisions de référence pour chacun des x' sont évidemment :

$$p'_t(x', s_{t-1}) = \sum_{x \in x'} p_t(x, s_{t-1}). \qquad (3.34)$$

Si p satisfait à (3.33), alors nous avons trivialement :

$$p'_{t+1}(x', s_{t-1}.r) = (1-\alpha_t)p'_t(x', s_{t-1}) + \begin{cases} \alpha_t, & \text{si } r \in x' \\ 0, & \text{si } r \notin x'. \end{cases}$$

Il en résulte que S' obéit également à un modèle de lissage de mêmes coefficients $\{\alpha_t\}$ que le modèle p.

Propriété 2.

Cette propriété est une particularisation de l'identité (3.7) du théorème 3.2 au cas où m est induit par un modèle explicite p. Dans ce cas, (3.7) devient :

$$p_{t+1}(z_t(c), S_t) = \min \{p_{t+1}(z_t(c-1), S_t), p_{t+1}(y_t(c), S_t)\}$$
$$\leq p_{t+1}(y_t(c), S_t), \text{ a fortiori.} \qquad (3.35)$$

Théorème 3.7.

Pour une chaîne de références obéissant à un modèle de lissage homogène p de coefficient α, posons : $\alpha' = 1-\alpha$. Sous l'algorithme induit M, nous avons les majorations :

$$f(c) = \lim \Pr[R_t \notin E_{t-1}(c, M)] \leq \alpha'^{c},$$

quels que soient c, α, et $p_1(.)$.

Démonstration.

Cette inégalité est triviale pour c = 1. En effet, en vertu de (3.33), la probabilité de référence d'une page qui vient d'être référencée vaut au moins α, de sorte que :

$$\Pr[R_t \in E_{t-1}(1, M)] = \Pr[R_t = R_{t-1}] \geq \alpha, \ \forall\ t.$$

Supposons-la vraie pour c-1, c'est-à-dire que :

$$f(c-1) \leq \alpha'^{c-1}.$$

Posons :

$$f_t(c) = \Pr[R_t \notin E_{t-1}(c, M)],$$

$$f'_t(c) = 1-f_t(c) = \sum_{s_{t-1}} \Pr[S_{t-1} = s_{t-1}]\, f'_t(c, s_{t-1}),$$

où :

$$f'_t(c, s_{t-1}) = \sum_{x \in E_{t-1}(c,M)} p_t(x, s_{t-1}) = P'_t(E_{t-1}(c,M), s_{t-1}).$$

Montrons tout d'abord que :

$$f'_{t+1}(c, s_{t-1}.r) \geq \alpha' f'_t(c-1, s_{t-1}) + \alpha, \ \forall\ c, r, (t, s_{t-1}). \tag{3.36}$$

Trois cas sont à distinguer, suivant la réalisation r de R_t.

1. $r \in E_{t-1}(c, M)$.

Dans ce cas : $E_t(c, M) = E_{t-1}(c, M)$, puisque M est à la demande; et nous avons

$$f'_{t+1}(c, s_{t-1}.r) = \alpha' f'_t(c, s_{t-1}) + \alpha$$
$$\geq \alpha' f'_t(c-1, s_{t-1}) + \alpha.$$

2. $r \notin E_{t-1}(c, M)$ et $|E_{t-1}(c, M)| < c$.

Dans ce cas : $E_t(c, M) = E_{t-1}(c, M) + r$, et nous avons :

$$f'_{t+1}(c, s_{t-1}.r) = \alpha' f'_t(c, s_{t-1}) + p_{t+1}(r, s_{t-1}.r)$$
$$\geq \alpha' f'_t(c-1, s_{t-1}) + \alpha.$$

3. $r \notin E_{t-1}(c, M)$ et $|E_{t-1}(c, M)| = c$.

Dans ce cas : $E_t(c, M) = E_{t-1}(c, M) + r - z_t(c)$, et :

$$f'_{t+1}(c, s_{t-1}.r) = \alpha' f'_t(c, s_{t-1}) + p_{t+1}(r, s_{t-1}.r)$$
$$- p_{t+1}(z_t(c), s_{t-1}.r).$$

Mais, en vertu de (3.35) :

$$p_{t+1}(z_t(c), s_{t-1}.r) \leq p_{t+1}(y_t(c), s_{t-1}.r)$$
$$= f'_{t+1}(c, s_{t-1}.r) - f'_{t+1}(c-1, s_{t-1}.r)$$
$$= \alpha'(f'_t(c, s_{t-1}) - f'_t(c-1, s_{t-1})).$$

La dernière expression de $f'_{t+1}(c, s_{t-1}.r)$ satisfait donc également à l'inégalité (3.36).

Cela étant, cette inégalité implique :

$$\Pr[R_{t+1} \in E_t(c, M) \mid S_{t-1}=s_{t-1}] = \sum_{r \in X} p_t(r, s_{t-1}) f'_{t+1}(c, s_{t-1}.r$$
$$\geq \alpha' f'_t(c-1, s_{t-1}) + \alpha,$$

et, "inconditionnellement" :

$$f'_{t+1}(c) = \Pr[R_{t+1} \in E_t(c, M)]$$
$$\geq \sum_{s_{t-1}} \Pr[S_{t-1} = s_{t-1}] \ (\alpha' f'_t(c-1, s_{t-1}) + \alpha)$$
$$= \alpha' f'_t(c-1) + \alpha.$$

D'où, pour $t \to \infty$:

$$f(c) = 1 - \lim f'_t(c)$$
$$= 1 - \alpha - \alpha' \lim f'_t(c-1)$$
$$\leq \alpha' \lim f_t(c-1)$$
$$\leq \alpha'^{c}, \quad \text{par induction.} \quad \text{C.Q.F.D.}$$

Si, en première approximation, nous identifions f(c) à sa borne supérieure α'^{c}, alors nous pouvons faire les remarques suivantes.

- Pour c fixé, un taux de défauts de page faible est obtenu pour de programmes à forte localisation (α grand).
- Pour un programme donné (α fixé), f(c) décroît exponentiellement avec c. D'autres approximations de f(c), basées sur certains résultats expérimentaux, ont été proposées par Belady et Kuehner [BeK69] et par Chamberlin, Fuller, et Liu [CFL73].
- Les résultats précédents permettent également une discussion de l'incidence de la taille des pages sur le taux de défauts. Supposons que, au niveau de la plus petite unité de code adressable (mot), le processus des références obéit à un modèle de lissage homogène de coefficient α. Alors, pour toute partition du code en pages, le processus des références de pages obéit aussi à un modèle de lissage homogène de coefficient α, en vertu de la propriété 1. D'autre part, lorsque le nombre Q de mots de mémoire centrale alloués au programme est fixé, la taille Z des pages est inversément proportionnelle au nombre c de cadres alloués : $Q = c.Z$. Avec ce modèle de comportement, il résulte donc que, pour Q fixé, le taux de défauts de page croît (exponentiellement) avec Z. A première vue, ce résultat ne choque pas

l'intuition : une partition fine (Z petit) permet des processus de prévision et d'allocation fins et précis également. Cette croissance du taux de défauts de page avec Z est confirmée, du moins pour de grandes valeurs de Z, par de nombreuses études expérimentales [KuL69, Hat72, LeB75]. Cependant, on observe que ce taux commence généralement par décroître avec Z. A notre avis, ce fait peut être interprété de la façon suivante : il existe aussi une affinité entre les informations voisines, et les pages ne doivent pas être trop petites sous peine de détruire cette affinité.

Le choix de la taille des pages doit également tenir compte de beaucoup d'autres facteurs tels que l'espace de mémoire perdu par arrondi à l'intérieur des pages (fragmentation interne), ou celui requis par les diverses tables de conversion d'adresses du système. Pour une analyse plus détaillée de ce problème, on pourra se référer à [KuL69, Hat72, GTB73, LeB75].

3.5.4. Optimalité de l'algorithme induit.

Pour des chaînes de références obéissant à un modèle de lissage exponentiel, nous allons montrer que l'algorithme induit, qui remplace la page la moins susceptible d'être référencée dans l'immédiat, est optimal, et cela quels que soient la taille de mémoire allouée au programme et l'horizon d'optimisation. Vu la grande diversité des structures probabilistes particulières qu'on peut obtenir selon les valeurs accordées aux coefficients de lissage, c'est un résultat assez puissant. Ainsi, il contient non seulement, comme cas particulier, l'optimalité de l'algorithme A_0 d'Aho et al pour des modèles à références indépendantes, mais permet également de conclure, par exemple, à l'optimalité de l'algorithme LFU dans le cas où les probabilités de référence de chaque page sont égales à leur fréquence d'utilisation passée.

Supposons désormais le modèle de lissage (3.33) vrai. Les résultats suivants s'appliquent pour tous :

t et T, $t \leq T$,

$c = 1, 2, \ldots,$

$e \subset X$, tel que $|e| = c$,

$r \notin e$,

$s_{t-1} \in X^{t-1}$.

Théorème 3.8.

L'algorithme induit est optimal. Autrement dit, nous avons

$$D\tilde{F} \overset{\text{déf}}{=} \tilde{F}_t(e+r-y, s_{t-1}) - \tilde{F}_t(e+r-z, s_{t-1}) \geq 0, \qquad (3.37)$$

pour z tel que

$$p_t(z, s_{t-1}) = \min_{x \in e} p_t(x, s_{t-1}), \qquad (3.38)$$

et pour tout y dans e.

Un premier lemme utilisé dans la démonstration de ce théorème a été établi dans /ADU71/. Il est d'ailleurs valable quelle que soit la distribution de probabilité Pr intrinsèque à S.

Lemme 3.4.

$$D\tilde{F} \leq 1, \text{ pour tout } y \in e. \tag{3.39}$$

Ensuite, si l'inégalité (3.37) est triviale pour t = T (auquel cas $D\tilde{F} = p_T(y, s_{T-1}) - p_T(z, s_{T-1}) \geq 0$), elle n'est pas suffisante pour pouvoir s'induire elle-même dans une démonstration par récurrence. Les deux lemmes suivants, dont la démonstration sera donnée en annexe, constituent un renforcement du théorème 3.8 dans le cas où $p_t(z, s_{t-1}) \leq p_t(r, s_{t-1})$.

Lemme 3.5.

Supposons l'inégalité (3.37) vraie pour T, T-1,..., t+1.

Soient y et z deux pages de e telles que :

$$p_t(z, s_{t-1}) = \min_{x \in e+r} p_t(x, s_{t-1}), \tag{3.40}$$

$$p_t(y, s_{t-1}) = \min_{x \in e+r-z} p_t(x, s_{t-1}). \tag{3.41}$$

Alors $D\tilde{F}$ peut se mettre sous la forme :

$$D\tilde{F} = a.\ \varphi_t(b), \tag{3.42}$$

où :

$$a = p_t(y, s_{t-1}) - p_t(z, s_{t-1}) \geq 0, \tag{3.43}$$

$$b = p'_t(e+r-y-z, s_{t-1}), \tag{3.44}$$

et où φ_t est définie récursivement par :

$$\varphi_T(b) = 1, \tag{3.45}$$

$$\varphi_t(b) = \alpha'_t b \varphi_{t+1}(\alpha'_t b + \alpha_t) + 1, \text{ pour } t < T \tag{3.46}$$

(nous avons posé $\alpha'_t = 1-\alpha_t$).

φ_t possède en outre la propriété suivante : quels que soient les réels non négatifs a, b, c, et d,

si $a+b = c+d \leq 1$ et $a \geq c$, alors $a\varphi_t(b) \geq c\varphi_t(d)$. (3.47)

Lemme 3.6.

Si l'inégalité (3.37) est vraie pour T, T-1,..., t+1, et si z satisfait à (3.40), alors :

$$D\tilde{F} \geq a.\ \varphi_t(b) \geq 0, \tag{3.48}$$

la fonction φ_t ayant été définie au lemme précédent.

Nous sommes maintenant en mesure de donner la

Démonstration du théorème 3.8.

L'inégalité (3.37) est triviale pour t = T. Supposons-la vraie pour t+1. Si z satisfait à (3.40), alors (3.37) est vraie à l'instant t, en vertu du lemme 3.6. Il reste donc à considérer le cas :

$$p_t(r, s_{t-1}) = \min_{x \in e+r} p_t(x, s_{t-1}). \tag{3.49}$$

Notons alors que, d'après la définition (3.38), la page z est, après r, celle de plus petite probabilité de référence dans e+r. Dans ce cas, la formule (3.9) donne, en vertu de l'hypothèse d'induction :

$$\begin{aligned} D\tilde{F} = & \sum_{r_t \in e+r-y-z} p_t(r_t, s_{t-1})\{\tilde{F}_{t+1}(e+r-y, s_{t-1}.r_t) - \tilde{F}_{t+1}(e+r-z, s_{t-1}.r_t)\} \\ & + \sum_{r_t \notin e+r} p_t(r_t, s_{t-1})\{\tilde{F}_{t+1}(e+r_t-y, s_{t-1}.r_t) - \tilde{F}_{t+1}(e+r_t-z, s_{t-1}.r_t)\} \\ & + p_t(y, s_{t-1})\{1+\tilde{F}_{t+1}(e, s_{t-1}.y) - \tilde{F}_{t+1}(e+r-z, s_{t-1}.y)\} \\ & + p_t(z, s_{t-1})\{\tilde{F}_{t+1}(e+r-y, s_{t-1}.z) - 1 - \tilde{F}_{t+1}(e, s_{t-1}.z)\}. \end{aligned}$$

Par induction, les termes correspondant au cas $r_t \neq y, z$ sont non négatifs, puisque $p_{t+1}(z, s_{t-1}.r_t) = \min_{x \in e} p_{t+1}(x, s_{t-1}.r_t)$ aussi.

Quant aux deux derniers termes (ddt), ils peuvent s'écrire :

$$ddt = a - p_t(y, s_{t-1})D_y + p_t(z, s_{t-1})D_z,$$

où a a été défini par (3.43), et où :

$$D_y = \tilde{F}_{t+1}(e+r-z, s_{t-1}\cdot y) - \tilde{F}_{t+1}(e, s_{t-1}\cdot y)$$

$= c'\varphi_{t+1}(d')$, en vertu du lemme 3.5, où nous avons posé

$$c' = p_{t+1}(z, s_{t-1}\cdot y) - p_{t+1}(r, s_{t-1}\cdot y)$$

$$= \alpha'_t(p_t(z, s_{t-1}) - p_t(r, s_{t-1})),$$

$$d' = P'_{t+1}(e-z, s_{t-1}\cdot y) = \alpha'_t P'_t(e-z, s_{t-1}) + \alpha_t;$$

$$D_z = \tilde{F}_{t+1}(e+r-y, s_{t-1}\cdot z) - \tilde{F}_{t+1}(e, s_{t-1}\cdot z)$$

$\geq a'\varphi_{t+1}(b')$, en vertu du lemme 3.6, où nous avons posé

$$a' = \alpha'_t(p_t(y, s_{t-1}) - p_t(r, s_{t-1})),$$

$$b' = \alpha'_t P'_t(e-y, s_{t-1}) + \alpha_t.$$

En vertu du lemme 3.5 (propriété (3.47)) :

$$a'\varphi_{t+1}(b') \geq c'\varphi_{t+1}(d').$$

D'où finallement :

$$ddt \geq a(1 - D_y)$$

≥ 0, en vertu du lemme 3.4. C.Q.F.D.

3.6. Conclusions.

En prenant comme objectif d'une pagination à partition fixe la minimisation du nombre de chargements de pages encourus pendant l'exécution d'un programme, nous avons ramené le problème d'allocation à un problème de remplacement. Dans ces quelques pages, nous espérons avoir explicité un aspect des relations qui devraient exister entre le comportement des programmes et l'algorithme de remplacement. Pour mathématiques qu'elles puissent être, la plupart de ces relations n'en sont pas moins conformes à l'intuition.

D'une façon heuristique, pour que le risque de défaut de page soit constamment faible, l'idéal serait évidemment d'avoir toujours en mémoire centrale les pages les plus susceptibles d'être prochainement référencées. Or, une pagination à la demande ne permet qu'une variation relativement lente des états-mémoire, deux états consécutifs ne différant, tout au plus, que de la page référencée. La CIL, qui implique une évolution semblable de la part des localités instantanées (§ 3.3), devrait être satisfaite pour que le programme ne subisse que de rares défauts de page. Au contraire, imaginons un programme qui éparpille ses références de façon telle que, fréquemment, les localités instantanées soient totalement distinctes d'un instant à l'autre. Rarement, les pages en mémoire seront aussi les plus probables, et les défauts de page, même sous le meilleur algorithme, risquent d'être tellement fréquents que la pagination même de tels programmes devrait être remise en question.

De façon à améliorer la localisation des programmes, une attention particulière doit être accordée au style de programmation même. Les expériences de Brawn et Gustavson /BrG68/ montrent même que, en ce qui concerne les performances des programmes dans une mémoire paginée, le style de programmation est souvent plus important que tout autre facteur, tel que l'architecture du système ou l'algorithme de remplacement. Divers conseils de programmation visant à développer des programmes à forte localisation ont également été formulés dans /RaK68, Say69, Mor73/. La localisation est également affectée par la façon dont les "modules" du programme sont groupés en pages. Le problème théorique du partitionnement (en anglais : pagination) optimal n'a été résolu que dans certains cas particuliers /Ker69/. Dans la pratique, on groupera, autant que possible, les modules par affinité /Com67, HaG71/.

Après avoir formulé ce que doit être un "bon" comportement de programme, et en supposant ce comportement connu, il reste à en déduire les principes d'un "bon" algorithme de pagination. Si les pages les plus probables à un instant donné le restent, en toute vraisemblance, à l'étape suivante, et ainsi de suite, alors, vu cette stationnarité, un objectif à court terme ne devrait pas contredire un objectif à long terme. De la sorte, une stratégie optimale à terme immédiat pourrait rester optimale, ou presque, quel que soit l'horizon d'optimisation.

Les principes d'un remplacement efficace sont beaucoup moins simples pour d'autres types de comportement de programme. Ainsi, pour des programmes obéissant à des modèles du type chaîne de Markov, la méthode itérative de Howard peut être utilisée pour déterminer les stratégies de remplacement optimales, pourvu que le nombre d'états (pages du programme) ne soit pas trop élevé /InK74/.

Signalons pour terminer quelques variantes du problème d'optimisation dynamique considéré dans ce chapitre. En prenant comme coût d'une pagination le nombre de chargements de pagès requis pendant l'exécution d'un programme, nous avons implicitement admis que les durées de transfert des pages entre les deux niveaux de la mémoire sont toujours les mêmes, pour toutes les pages.

Une généralisation possible consiste à considérer la durée de transfert d'une page comme dépendante de l'emplacement de la page sur la mémoire auxiliaire par rapport à la position actuelle de la tête de lecture/écriture. Les décisions de remplacement doivent alors tenir compte également de ces positions relatives /GLP74/. Le problème de l'arrangement des pages sur une mémoire auxiliaire à bulles magnétiques, de façon à minimiser les mouvements de la tête de lecture, a été traité dans /Mit74/.

Lorsqu'il est plus avantageux de charger plusieurs pages d'un coup que de les charger une à une, certains algorithmes à préchargement (prepaging) méritent d'être considérés /Spa74/.

3.7. Annexe : démonstration des lemmes.

Démonstration du lemme 3.2.

Ce lemme découle directement du lemme suivant :

Lemme 3.2-bis.

Sous les conditions C2 et C3, les permutations β_k sont

$$\beta_k(i) = \begin{cases} i & \text{pour } i \prec \beta_k(k), \\ i+1 & \text{pour } \beta_k(k) \preccurlyeq i \prec k, \\ \beta_k(k) & \text{pour } i = k, \\ i & \text{pour } i \succ k. \end{cases}$$

En effet, si $i \prec j$, alors le lemme 3.2-bis implique

$\beta_j(i) \preccurlyeq i+1$, et

$\beta_i(j) = j$.

Ceci achève la démonstration du lemme 3.2.

Démonstration du lemme 3.2-bis.

β_k étant une permutation de $\{1,\ldots,n\}$, nous avons pour tout i:

$\beta_k(i) = i + i_+ - i_-$,

où i_+ est le nombre d'indices j supérieurs à i mais dont l'image $\beta_k(j)$ est inférieure à celle de i :

$$i_+ = \left|\{j \mid j \succ i \text{ et } \beta_k(j) \prec \beta_k(i)\}\right|,$$

et où :

$$i_- = \left|\{j \mid j \prec i \text{ et } \beta_k(j) \succ \beta_k(i)\}\right|.$$

Le lemme étant trivial pour $i = k$, examinons le cas $i \neq k$.

Dans ce cas, i_- est toujours nul :

$i_- = 0$.

En effet, en vertu de C3, il ne peut exister tout au plus qu'un seul élément j satisfaisant à la condition "$j \prec i$ et $\beta_k(j) \succ \beta_k(i)$" à savoir $j = k$. Par conséquent : $i_- \leq 1$.

Supposons, par l'absurde, que $i_- = 1$. k satisfait donc à : $k \prec i$ et $\beta_k(k) \succ \beta_k(i)$. Or, l'inégalité $\beta_k(k) \succ \beta_k(i)$ équivaut à : $\beta_k(i) \leq \beta_k(k)-1$. D'autre part, en vertu de C2 : $\beta_k(k) \leq k \prec i$, c'est-à-dire : $\beta_k(k) \leq i-1$. Par conséquent : $\beta_k(i) \leq i-2$, et : $i_- \geq 2$, ce qui contredit $i_- = 1$.

Notons que l'égalité $i_- = 0$ qui vient d'être démontrée implique la CIL :

$$\beta_k(i) \geq i, \forall\ k, i \text{ tels que } i \neq k.$$

Il reste à démontrer que, pour $i \neq k$:

$$i_+ = \begin{cases} 1 & \text{pour } \beta_k(k) \leq i \prec k, \\ 0 & \text{dans le cas contraire.} \end{cases}$$

En utilisant C3, on peut montrer, comme précédemment, que $i_+ \leq 1$, et que $i_+ = 1$ si et seulement si $k \succ i$ et $\beta_k(k) \prec \beta_k(i)$. Cette dernière condition équivaut précisément à $\beta_k(k) \leq i \prec k$, puisque $\beta_k(i) = i + i_+ - i_- = i + 1 - 0 = i+1$. C.Q.F.D.

Démonstration du lemme 3.3.

En vertu de (3.27), l'ensemble $N_{t-1} = m_t^{-1}(E_{t-1}(c, M), S_{t-1})$ des rangs des pages dans $E_{t-1}(c, M)$ est de la forme :

$$N_{t-1} = C + I,$$

où $C = \{1,\ldots,c-1\}$, et où I est soit $\emptyset$, soit réduit à un seul élément $i \geq c$.

Montrons d'abord que $N_t = m_{t+1}^{-1}(E_t(c, M), S_t)$ vaut :

$$N_t = \begin{cases} C + I, \text{ si } k \prec c, \\ \beta_k(C) + \beta_k(k), \text{ dans le cas contraire.} \end{cases} \qquad (3.50)$$

En effet :

1. $k \prec c$.

Il n'y a donc pas de défaut de page, et $E_t(c, M) = E_{t-1}(c, M)$.

Nous avons alors :

$$N_t = \beta_k(N_{t-1}) = \beta_k(C) + \beta_k(I) = C + I.$$

En effet :

— L'identité entre les ensembles C et $\beta_k(C)$, de même cardinal, peut être établie de la façon suivante. S'il existait un élément $j \in C - \beta_k(C)$, alors nous avons simultanément

$$j \prec c, \text{ et } j = \beta_k(j'), \text{ pour un certain } j' \geq c.$$

Or $k \prec c$, donc $j' \neq k$, et, en vertu de la CIL, $\beta_k(j') \geq j'$.

Par conséquent :

$$c \succ j = \beta_k(j') \geq j' \geq c,$$

ce qui est absurde.

— D'autre part, pour $i \geq c \succ k$:

$$\beta_k(i) = i,$$

en vertu du lemme 3.2-bis.

2. $k \geq c$.

2.1. Si $I = \emptyset$, alors il y a défaut de page mais ne nécessitant pas de remplacement. La page $m_t^{-1}(k, S_{t-1})$ est chargée et son rang devient $\beta_k(k)$.

2.2. Si $I = \{i\}$, avec $i \geq c$, alors 2 sous-cas sont à distinguer.

2.2.1. Ou bien $k = i$, alors il n'y a pas de défaut, et :

$$N_t = \beta_k(N_{t-1}) = \beta_k(C) + \beta_k(k).$$

2.2.2. Ou bien $k \neq i$, et il y a un défaut de page requérant un remplacement. En vertu de C3, la page $m_t^{-1}(i, S_{t-1})$ reste celle de rang le plus petit, et est donc remplacée.

Ceci achève la démonstration de (3.50). De cette égalité, il résulte que, pour tout $i \geq c$:

$$i \in N_t \Leftrightarrow \text{ ou } \begin{cases} i \in N_{t-1} \text{ et } k \prec c, \\ i \in \beta_k(C) + \beta_k(k) \text{ et } k \geq c. \end{cases}$$

Particularisons la condition " $i \in \beta_k(C) + \beta_k(k)$ et $k \geq c$ " au cas où $i \succ c$. Dans ce cas, il est impossible que $i \in \beta_k(C)$. En effet, pour tout $j \prec c$, et pour tout k :

$$\beta_k(j) = j+1 \preceq c,$$

en vertu du lemme 3.2-bis. Donc $i \neq \beta_k(j)$, pour tout $j \in C$. La condition examinée équivaut alors à " $i = \beta_k(k)$ et $k \geq c$ ". Il en résulte alors l'équivalence (3.30).

Quant à la condition " $c \in \beta_k(C) + \beta_k(k)$ et $k \geq c$ ", elle équivaut à " $\beta_k(k) \preceq c \preceq k$ ", en conformité avec (3.29) (en effet, même si $\beta_k(k) \prec c$, nous avons $c = \beta_k(c-1) \in \beta_k(C)$). C.Q.F.D.

Démonstration du lemme 3.5.

Notons tout d'abord que supposer que l'inégalité (3.37) est vraie pour t+1,..., T, implique que, à l'instant t+1, l'algorithme optimal doit remplacer la page de plus petite probabilité de référence. Cela étant, commençons par établir la forme (3.42) de $D\tilde{F}$, par induction. Les équations fonctionnelles (3.8) donnent, compte tenus de (3.40) et (3.41) :

$$D\tilde{F} = \sum_{r_t \in e+r-y-z} p_t(r_t, s_{t-1})\{\tilde{F}_{t+1}(e+r-y, s_{t-1}.r_t) - \tilde{F}_{t+1}(e+r-z, s_{t-1}.r_t)\}$$
$$+ p_t(y, s_{t-1}) - p_t(z, s_{t-1}).$$

Or, pour $r_t \in e+r-y-z$, nous avons, en vertu de (3.33) :

$$p_{t+1}(z, s_{t-1}.r_t) = \alpha_t' p_t(z, s_{t-1})$$
$$= \min_{x \in e+r} p_{t+1}(x, s_{t-1}.r_t),$$
$$p_{t+1}(y, s_{t-1}.r_t) = \alpha_t' p_t(y, s_{t-1})$$
$$= \min_{x \in e+r-y} p_{t+1}(x, s_{t-1}.r_t),$$
$$p'_{t+1}(e+r-y-z, s_{t-1}.r_t) = \alpha_t' p'_t(e+r-y-z, s_{t-1}) + \alpha_t.$$

Par induction, chaque terme entre accollades vaut

$$\tilde{F}_{t+1}(e+r-y, s_{t-1}.r_t) - \tilde{F}_{t+1}(e+r-z, s_{t-1}.r_t) = \alpha_t' a \varphi_{t+1}(\alpha_t' b + \alpha_t),$$

avec a et b définis par (3.43) et (3.44). Par conséquent

$$D\tilde{F} = b.\alpha_t' a \varphi_{t+1}(\alpha_t' b + \alpha_t) + a$$
$$= a \varphi_t(b),$$

où nous avons posé

$$\varphi_t(b) = \alpha_t' b \varphi_{t+1}(\alpha_t' b + \alpha_t) + 1,$$

conformément à (3.46).

Pour établir (3.47), nous utiliserons la propriété suivante :

$$a+b \leq 1 \Rightarrow a\varphi_t(b) \leq 1. \qquad (3.51)$$

Cette propriété, qu'on peut rapprocher du lemme 3.4, peut se vérifier facilement par induction. En effet :

$$a\varphi_t(b) = a + b\alpha_t' a\varphi_{t+1}(\alpha_t' b + \alpha_t), \text{ en vertu de (3.46)}$$
$$\leq a + b, \text{ par induction}$$
$$\leq 1.$$

La démonstration de (3.47) se fait également par induction. En vertu de (3.46) :

$$a\varphi_t(b) - c\varphi_t(d) = a(\alpha_t' b\varphi_{t+1}(\alpha_t' b + \alpha_t) + 1) - c(\alpha_t' d\varphi_{t+1}(\alpha_t' d + \alpha_t) + 1)$$
$$= (a-c)(1 - \alpha_t' c\varphi_{t+1}(\alpha_t' d + \alpha_t))$$
$$+ b(\alpha_t' a\varphi_{t+1}(\alpha_t' b + \alpha_t) - \alpha_t' c\varphi_{t+1}(\alpha_t' d + \alpha_t)) .$$
$$(\text{puisque } d = (a-c) + b).$$

Le premier terme est non négatif, en vertu de (3.51), et le second est aussi non négatif, par induction. C.Q.F.D.

Démonstration du lemme 3.6.

Sous les hypothèses du lemme, nous avons :

$$D\tilde{F} = \sum_{r_t \in e+r-y-z} p_t(r_t, s_{t-1})\{\tilde{F}_{t+1}(e+r-y, s_{t-1}.r_t) - \tilde{F}_{t+1}(e+r-z, s_{t-1}.r_t)\}$$
$$+ p_t(y, s_{t-1}) + p_t(z, s_{t-1})(D_z - 1)$$
$$+ \sum_{r_t \notin e+r} p_t(r_t, s_{t-1}).\ D_{r_t},$$

où

$$D_z \overset{\text{déf}}{=} \tilde{F}_{t+1}(e+r-y, s_{t-1}.z) - \min_{z' \in e+r-z} \tilde{F}_{t+1}(e+r-z', s_{t-1}.z)$$
$$\geq 0, \text{ en vertu du choix non nécessairement optimal } z' = y;$$

$$D_{r_t} = \tilde{F}_{t+1}(e+r-y+r_t-z, s_{t-1}.r_t) - \min_{z' - e+r-z} \tilde{F}_{t+1}(e+r-z+r_t-z', s_{t-1}.r_t)$$
$$\geq 0, \text{ un choix possible pour } z' \text{ étant } y.$$

En vertu de ces deux dernières inégalités, $D\tilde{F}$ majore une expression qui est précisément la valeur de $D\tilde{F}$ dans le cas du lemme précédent. On pourra invoquer des arguments similaires à ceux du lemme 3.5 pour terminer la démonstration du lemme 3.6. C.Q.F.D.

4. ALLOCATION A PARTITION VARIABLE

4.1. Présentation du chapitre.

Par opposition à l'allocation à partition fixe examinée au chapitre précédent, sous une allocation à partition variable, aucune limite fixe n'est imposée quant à la quantité d'espace allouée à chacun des programmes. Moins contraignante, un tel mode d'allocation permet de gérer la mémoire centrale comme un pot commun dans lequel chaque programme peut puiser selon ses besoins propres et instantanés. Le paragraphe 4.2 présente une stratégie d'allocation comme des règles décisionnelles permettant de générer, tout au long de l'exécution de chacun des programmes, une séquence d'ensembles résidents (pages en mémoire centrale). La taille de ces ensembles n'est pas contrainte par une borne supérieure; néanmoins, à côté du coût temporel qu'un programme encourt suite aux attentes de transferts de pages, nous considérons également un coût spatial mesurant la quantité d'espace allouée au programme. Cette pénalisation de l'occupation spatiale d'un programme peut se justifier par le fait que, la mémoire totale étant limitée, l'espace alloué à un programme l'est toujours au détriment des autres programmes.

Une allocation "efficace" devrait donc réaliser un certain compromis entre ces deux objectifs, contradictoires à première vue. Une façon de concilier ces deux critères a été proposé par Denning sous le nom de principe de l'ensemble d'activité [Den68a, Den68b]. Conceptuellement, c'est un principe de l'offre selon les besoins. A tout moment de l'exécution d'un programme, on doit

allouer à ce dernier toutes ses pages qui sont très vraisemblablement requises dans un futur proche, et ces pages-là seulement. D'un point de vue pratique, une telle allocation exige que le système d'exploitation dispose d'une certaine méthode pour estimer les besoins instantanés des programmes sur la base de leur comportement passé. La plupart de ces méthodes se basent sur les références récentes des programmes pour estimer leur besoin actuel, tenant compte ainsi de la localisation des références. Ainsi, l'ensemble d'activité temporel (time-window working set) /Den68a, Den68b/ d'un programme est constitué, à chaque instant, de ses pages qui ont été référencées au moins une fois depuis une certaine durée fixe de temps virtuel. Les propriétés et les performances des stratégies à l'ensemble d'activité temporel seront analysées au paragraphe 4.3, à travers, notamment, certains modèles de comportement des chapitres précédents.

Le paragraphe 4.4 formalise diverses méthodes pour comparer des politiques d'allocation sur la base à la fois de leur coût spatial et de leur coût temporel (comparaison bi-critère). Chacune de ces méthodes est définie par une structure de domination /Yu72, Yu73/ spécifiant dans quels cas une politique sera préférée à une autre. Une structure de domination particulière réflète la quantité et la précision des informations dont l'analyste de système dispose pour quantifier l'importance relative de chacune des deux fonctions de coût.

Les stratégies à l'ensemble d'activité potentiel (probability working set) sont définies au paragraphe 4.5. Sous ce genre de stratégies, une page de programme ne doit résider en mémoire

centrale à un instant donné que si elle est actuellement requise par le processeur ou si sa probabilité d'être référencée dans l'immédiat excède un certain niveau fixé. Des exemples de ces stratégies sont ensuite analysés pour divers modèles de comportement de programme. Nous établirons enfin que, pour diverses structures de domination envisagées au paragraphe précédent, les stratégies à l'ensemble d'activité potentiel sont des stratégies non dominées : il n'existe pas d'autres stratégies qui puissent leur être préférées.

Signalons que ce genre d'optimalité a été pressentie par Denning lorsqu'il énonçait la non domination (au sens de Pareto) des stratégies à l'ensemble d'activité temporel pour des programmes exhibant une certaine localisation dans leur comportement [Den68b, pp. 73-79]. Cette proposition peut être réfutée par un contre-exemple simple (modèle à liste lru). Récemment, l'analyse bi-critère des stratégies d'allocation a été abordée dans le cas déterministe (algorithme GOPT [DeS76], algorithme VMIN [PrF76]). Nous nous limiterons essentiellement au cas d'un avenir non déterministe [TQT75c, DeS76, DeT76, TQT76b].

4.2. Formulation du problème de l'allocation de la mémoire.

En toute généralité, les méthodes permettant une utilisation dynamique et partagée de la mémoire centrale d'un système de multiprogrammation à mémoire virtuelle sont à la fois difficiles à formuler et à analyser. Il s'agit de déterminer, à tout instant (réel), les transferts de pages à entreprendre, et cela en vue d'optimiser certaines mesures de la performance globale du système. Il est alors évident que, l'allocation de la mémoire centrale, en tant que fonction particulière du système d'exploitation en général, doit être conçue "en concordance" avec les autres fonctions telles que la gestion du processeur ou celle des mémoires périphériques de pagination. On est alors amené à modéliser le système comme un réseau de files d'attente plus ou moins complexes. La résolution de ce genre de modèles peut permettre de dégager certains aspects ayant trait à la coordination et aux interactions entre les diverses composantes du système [Buz71, Cou71, Gel73b, Bra74, BGLP75, LeP76, DeK76, ...].

Notre problème est quelque peu différent. Nous voulons mettre en évidence les relations qui devraient exister entre l'allocation de la mémoire et le comportement intrinsèque des programmes. En d'autres termes, à partir d'une certaine connaissance des besoins en espace des programmes, il s'agit de déterminer les principes d'une allocation raisonnable et efficace.

La formulation suivante s'inspire notamment des premiers travaux de Denning [Den68a, Den68b]. Fixons notre attention sur un certain programme caractérisé par une chaîne de références :

$$S = R_1 \ldots R_t \ldots ; \quad t = 1, 2, \ldots$$

En ce qui concerne ce programme, une stratégie d'allocation de la mémoire centrale peut être décrite par la séquence des états instantanés de la mémoire que cette stratégie génère :

$$A = A_0.A_1...A_t...; \; t = 1, 2, ... \tag{4.1}$$

A_t représente l'ensemble des pages du programme pour lesquelles un cadre de mémoire centrale est alloué, lors de la $t^{ième}$ opération élémentaire du programme. N'imposant pas de limite au nombre de cadres que le programme peut occuper, nous exigerons simplement que :

$$A_t \ni R_t, \; \forall \; t \succ 0. \tag{4.2}$$

Comme coûts d'une stratégie d'allocation (générant une séquence d'ensemble résidents) A, nous considérons d'abord le nombre de chargements de pages encourus jusqu'à un certain instant T :

$$F_T(A) = \sum_{t=0}^{T-1} f_t(A), \tag{4.3}$$

où :

$$f_t(A) = \left|A_{t+1} - A_t\right|, \; \forall \; t \geq 0. \tag{4.4}$$

Le coût spatial mesure la quantité totale d'espace alloué au programme :

$$G_T(A) = \sum_{t=0}^{T-1} g_t(A), \tag{4.5}$$

où :

$$g_t(A) = \left|A_t\right|, \; \forall \; t \geq 0. \tag{4.6}$$

Avec ces fonctions de coûts, comme dans le contexte d'une allocation à partition fixe, nous pouvons énoncer l'optimalité des stratégies d'allocation à la demande, sous lesquelles les pages ne sont chargées en mémoire centrale que lorsqu'elles sont requises par le processeur et jamais anticipativement. Dans ce

chapitre, nous définirons ces stratégies de la façon suivante :

Définition 4.1.

Une stratégie d'allocation A est à la demande si

$$A_{t+1} - A_t \subset R_{t+1}, \forall\, t \geq 0. \tag{4.7}$$

Evidemment, sous les stratégies à la demande, le nombre de chargements de pages est identique au nombre de défauts de pages :

$$f_t(A) = \begin{cases} 1, \text{ si } R_{t+1} \notin A_t \\ 0, \text{ dans le cas contraire} \end{cases}, \forall\, t \geq 0. \tag{4.8}$$

Théorème 4.1.

Pour toute stratégie d'allocation A^1, il existe une stratégie à la demande A^2, qui commence avec le même "état initial" :

$$A_0^2 = A_0^1, \tag{4.9}$$

et qui est uniformément meilleure que A^1:

$$F_T(A^2) \leq F_T(A^1), \forall\, T \geq 0, \tag{4.10}$$

et

$$G_T(A^2) \leq G_T(A^1), \forall\, T \geq 0. \tag{4.11}$$

Ce théorème est intuitivement évident et sera démontré en annexe. Il nous permet de limiter nos recherches aux stratégies à la demande. Notons que, sous ces stratégies, en vertu de (4.7) et de la contrainte d'allocation (4.2), deux états consécutifs de la mémoire sont liés par une relation de la forme :

$$A_{t+1} = A_t \cup R_{t+1} - Z_t, \tag{4.12}$$

où Z_t désigne un certain sous-ensemble de pages à libérer de A_t. Denning a proposé de libérer les pages n'ayant plus été utilisées depuis une certaine durée fixe de temps virtuel. C'est la stratégie à l'ensemble d'activité temporel [Den68a, Den68b] qui est implémentée, à quelques variantes près, sur de nombreux systèmes.

4.3. Stratégies à l'ensemble d'activité temporel.

Nous commençons ce paragraphe par rappeler et justifier les principes de l'allocation à l'ensemble d'activité temporel. Puis, nous présenterons diverses propriétés des coûts moyens spatial et temporel encourus sous ce genre de stratégies. Un premier résultat, dû à Denning et Schwartz [DeS72], permet d'exprimer ces coûts moyens en fonction de la distribution de probabilité de l'inter-référence des pages. Une propriété de convexité du taux de défauts de page, énoncée par Denning pour des programmes localisés [Den68b, p. 607], sera établie rigoureusement. Nous renvoyons le lecteur aux travaux de Lenfant [Len74a] pour une étude plus détaillée de la distribution de probabilité de la taille de l'ensemble d'activité temporel.

4.3.1. Principes de l'ensemble d'activité temporel.

Définition 4.2.

Soit $S = R_1 \ldots R_t \ldots$ la chaîne de références d'un programme. A l'instant virtuel t, son ensemble d'activité temporel de fenêtre arrière h, h étant un paramètre entier positif, est constitué des pages ayant été référencées au moins une fois durant les h dernières références du programme :

$$TWS_t(h) = \bigcup_{v=t-h+1}^{t} R_v. \qquad (4.13)$$

Une stratégie d'allocation A est une stratégie à l'ensemble d'activité temporel de fenêtre h si, pour tout t :

$$A_t = TWS_t(h). \qquad (4.14)$$

Cette stratégie sera notée TWS(h).

Les quelques commentaires suivants peuvent aider à mieux expliciter certains aspects du fonctionnement de ces stratégies.

1) L'équation (4.14) exprime notamment qu'un programme ne peut effectuer une opération élémentaire que si l'entièreté de son ensemble d'activité temporel réside actuellement en mémoire. Autrement dit, en termes de gestion du processeur, ce dernier ne peut jamais être alloué qu'aux programmes ayant tout leur ensemble d'activité temporel en mémoire. Cette exigence peut être interprétée comme une exigence de sécurité. Les pages utilisées dans un passé récent, estimées comme probablement réutilisées dans un futur immédiat, devraient donc résider en mémoire centrale afin d'assurer au programme un déroulement sans d'excessifs défauts de pages.

2) Ces stratégies sont des stratégies à la demande, au sens de la définition 4.1. Les pages n'appartenant à aucun ensemble d'activité temporel des programmes actifs peuvent être libérées, en ce sens qu'elles sont éligibles pour être remplacées. Physiquement, ces pages peuvent parfaitement être présentes en mémoire, bien qu'aucun cadre ne leur soit effectivement alloué.

3) Pour chaque programme, les ensembles résidents instantanés sont déterminés sur la base de leur comportement intrinsèque passé uniquement. On peut dire que l'allocation d'espace à chacun des programmes se fait d'une façon indépendante de celle des autres programmes.

4) Les besoins instantanés des programmes en pages sont estimés par leur besoins effectifs dans un passé récent. C'est une méthode de prévision adaptative justifiée par la localisation des référen-

ces. Des généralisations de cette règle "de la fenêtre arrière" ont été considérées récemment par Denning et Slutz /DeS76/. Nous les formulerons, à partir de modèles explicites de comportement de programme, au paragraphe 4.5.

4.3.2. Evaluation des coûts encourus sous les stratégies TWS(h).

Depuis les travaux originaux de Denning /Den68a, Den68b/, l'évaluation des coûts moyens :

$$\delta_t(h) = E\left[f_t(TWS(h))\right] = Pr\left[R_{t+1} \notin TWS_t(h)\right] \qquad (4.15)$$

et

$$\sigma_t(h) = E\left[g_t(TWS(h))\right] = E\left[|TWS_t(h)|\right] \qquad (4.16)$$

a fait l'objet de nombreuses recherches.

Tout d'abord, une relation entre ces deux coûts peut être obtenue à partir de la relation déterministe suivante :

$$TWS_t(h) = TWS_{t-1}(h-1) \cup \begin{cases} R_t, \text{ si } R_t \notin TWS_{t-1}(h-1) \\ \emptyset, \text{ dans le cas contraire.} \end{cases} \qquad (4.17)$$

Il en résulte alors que la taille de $TWS_t(h)$ peut se mettre sous la forme :

$$\begin{aligned} |TWS_t(h)| &= |TWS_{t-1}(h-1)| + B_{t-1}(h-1) \\ &= \sum_{i=1}^{h} B_{t-i}(h-i), \end{aligned} \qquad (4.18)$$

où les $B_{t-i}(h-i)$ désignent les variables booléennes :

$$B_{t-i}(h-i) = \begin{cases} 1, \text{ si } R_{t-i+1} \notin TWS_{t-i}(h-i) \\ 0, \text{ dans le cas contraire.} \end{cases} \qquad (4.19)$$

En prenant alors les espérances mathématiques des deux membres de (4.18), nous obtenons la relation suivante :

$$\sigma_t(h) = \sum_{i=1}^{h} \delta_{t-i}(h-i). \qquad (4.20)$$

Denning et Schwartz /DeS72/ ont exprimé ces coûts en fonction de la distribution de probabilité de l'inter-référence des pages (intervalle de temps entre deux références successives à une même page). Un raisonnement intuitif peut être le suivant. D'après la définition 4.2, une page référencée à un instant donné n'appartient pas à l'ensemble d'activité de fenêtre h si et seulement si elle n'a plus été référencée depuis au moins h références, c'est-à-dire si son inter-référence excède h. Les hypothèses probabilistes de /DeS72/ permettent d'obtenir la distribution de l'inter-référence en régime permanent. La chaîne de références S est supposée être une chaîne (au sens de la théorie des processus stochastiques) homogène dans le temps et récurrente non nulle :

$$\Pr[R_{t+k} \neq x \text{ pour } k=1,\ldots,j-1, \text{ et } R_{t+j}=x \mid R_t=x] = \rho_x(j), \quad \forall\, x \text{ et } \forall\, j, \text{ indépendamment de } t; \tag{4.21}$$

$$\sum_{j \geq 1} \rho_x(j) = 1, \quad \forall\, x; \tag{4.22}$$

$$\sum_{j \geq 1} j\rho_x(j) = \bar{r}_x < \infty, \quad \forall\, x. \tag{4.23}$$

Sous ces hypothèses, la distribution asymptôtique de l'inter-référence est donnée par :

$$\rho(j) = \lim_{t \to \infty} \Pr[R_{t+k} \neq R_t \text{ pour } k=1,\ldots,j-1, \text{ et } R_{t+j}=R_t]$$

$$= \sum_{x \in X} \rho_x(j)/\bar{r}_x, \quad \forall\, j, \tag{4.24}$$

et nous avons le

Théorème 4.2. /DeS72/

Pour des chaînes satisfaisant aux conditions (4.21-23), nous avons, lorsque $t \to \infty$, et pour tout h :

$$\delta_t(h) \to \delta(h) = \sum_{j \geq h} \rho(j), \text{ et} \tag{4.25}$$

$$\sigma_t(h) \to \sigma(h) = \sum_{i=1}^{h} \delta(h-i). \tag{4.26}$$

Ce théorème peut s'appliquer à un grand nombre de modèles de comportement, notamment aux modèles à références indépendantes et à liste lru pour lesquels (4.21-23) sont vérifiées.

Notons que, en vertu de (4.25), si la séquence $\rho(j)$ décroît avec j, alors $\delta(h)$ est une fonction convexe de h. Intuitivement, une telle convexité provient de la localisation des références : on s'attend en effet à ce que les pages qui n'ont plus été référencées depuis longtemps deviennent deviennent de moins en moins susceptibles de l'être à nouveau [Den68b, p. 60].
Cette intuition est formalisée par le

Théorème 4.3.

Pour des chaînes de références obéissant à un modèle qui satisfait à la CEL (définition 2.5), la différence

$$\Delta\delta_t(h) \stackrel{\text{déf}}{=} \delta_t(h) - \delta_t(h-1) \tag{4.27}$$

satisfait à l'inégalité :

$$\Delta\delta_t(h) - \Delta\delta_{t-1}(h-1) \geq 0, \quad \forall\ t, h. \tag{4.28}$$

Ce théorème implique évidemment que, si les $\delta(h) = \lim \delta_t(h)$ existent, alors les différences secondes de $\delta(h)$ sont non négatives, d'où la convexité de $\delta(h)$.

Démonstration.

Partons de l'équivalence triviale :

$R_{t+1} \notin TWS_t(h-1)$ si et seulement si

$$\begin{cases} \text{ou bien : } R_{t+1} \notin TWS_t(h-1) \text{ et } R_{t+1}=R_{t-h+1}, \\ \text{ou bien : } R_{t+1} \notin TWS_t(h). \end{cases}$$

Les différences premières de $\delta_t(h)$ valent donc :

$$\Delta\delta_t(h) = - \Pr[R_{t+1} \notin TWS_t(h-1),\ R_{t+1}=R_{t-h+1}]$$

$$= - \sum_{x \in X} \Pr[R_{t-h+1}=x,\ TWS_{t-1}(h-2) \not\ni x,\ R_t \neq x,\ R_{t+1}=x]$$

$$= - \sum_{x \in X} \sum_{s_{t-1} \in S^x_{t-1}} \Pr[S_{t-1}=s_{t-1}] . \Pr[R_t \neq x,\ R_{t+1}=x \mid s_{t-1}],$$

où S^x_{t-1} désigne le sous-ensemble suivant de X^{t-1} :

$$S^x_{t-1} = \{ s_{t-1} = R_1 \ldots R_{t-1} \mid R_{t-h+1}=x \text{ et } TWS_{t-1}(h-2) \not\ni x \}.$$

Mais, pour tout s_{t-1} et pour tout x :

$$\Pr[R_t \neq x,\ R_{t+1}=x \mid s_{t-1}] = \sum_{y \neq x} p_t(y, s_{t-1}) . p_{t+1}(x, s_{t-1} . y)$$

$$\leq p_t(x, s_{t-1}), \text{ en vertu de la CEL.}$$

Par conséquent :

$$\Delta\delta_t(h) \geq - \sum_{x \in X} \sum_{s_{t-1} \in S^x_{t-1}} \Pr[S_{t-1}=s_{t-1}] . p_t(x, s_{t-1})$$

$$= - \Pr[R_t \notin TWS_{t-1}(h-2),\ R_t=R_{t-h+1}], \text{ par définition de } S^x_{t-1}$$

$$= \Delta\delta_{t-1}(h-1). \quad \text{C.Q.F.D.}$$

A propos de cette propriété, signalons une célèbre conjecture de Denning [Den68b, pp 73-79] :

> Si $\delta(h)$ est convexe, alors les stratégies TWS(h) sont meilleures que toute autre stratégie : à coût spatial égal, elles génèrent le moins de défauts de page.

Une réponse à cette proposition sera donnée à la fin du § 4.5.3. Au stade actuel de l'exposé, l'analyse critique d'une proposition de ce genre peut inspirer les démarches suivantes :

1) Définition de méthodes permettant de comparer des stratégies sur la base des deux coûts F et G (→ § 4.4).

2) Recherche des "meilleures" stratégies possibles (→ § 4.5).

4.4. Comparaison entre stratégies d'allocation.

Au paragraphe 4.2, nous avons associé à tout processus d'allocation deux fonctions de coût, l'une, F, mesurant le nombre de chargements de pages, l'autre, G, pénalisant l'occupation spatiale. Un problème essentiel dans l'étude des stratégies d'allocation (à partition variable) est de se définir des méthodes qui permettent la comparaison de différentes stratégies, sur la base de ces deux fonctions de coûts.

On peut évidemment envisager un seul coût global, défini comme étant la somme (ou plus généralement, une combinaison linéaire à coefficients connus) des deux coûts F et G; entre plusieurs stratégies, on préférera celle conduisant au plus petit coût global. Cette méthode de comparaison, uni-critère, a été considérée notamment par Prieve et Fabry dans l'analyse, dans un contexte déterministe, de leur stratégie VMIN /PrF76/. Dans la pratique, l'utilisation d'une telle méthode de comparaison n'est pas toujours possible. En effet, même si l'on se limite à des mesures de l'efficacité d'une allocation qui sont des combinaisons linéaires de F et G, les coefficients d'une telle combinaison ne peuvent pas toujours être quantifiés avec précision : ce sont des paramètres faisant intervenir l'architecture logique du système aussi bien que des caractéristiques technologiques du matériel, et pour lesquels on ne dispose pas de moyens analytiques ou expérimentaux d'évaluation.

Pour représenter ce genre de situations où un critère de comparaison unique peut ne pas être spécifié d'une façon précise, nous ferons appel au formalisme des structures de domination

introduit par Yu pour des problèmes de comparaison multi-critères [Yu73].

4.4.1. Stratégies non dominées.

Considérons deux stratégies d'allocation A^1 et A^2 qui opèrent sur une même chaîne de références. Pour alléger les notations, nous désignerons leurs coûts moyens par :

$$(\bar{F}^1, \bar{G}^1) = (E[F_T(A^1)], E[G_T(A^1)]), \tag{4.29}$$

et

$$(\bar{F}^2, \bar{G}^2) = (E[F_T(A^2)], E[G_T(A^2)]) \tag{4.30}$$

respectivement.

Soit d_{12} le vecteur différence :

$$d_{12} = (\bar{F}^1 - \bar{F}^2, \bar{G}^1 - \bar{G}^2). \tag{4.31}$$

Définition 4.2.

> Une méthode de comparaison est définie lorsque, pour tout couple de stratégies A^1 et A^2, on se spécifie les valeurs de d_{12} pour lesquelles A^1 sera préférée à A^2. L'ensemble de ces valeurs, noté D, est appelé la structure de domination de la méthode de comparaison.
>
> Lorsque $d_{12} \in D$, on dit aussi que A^1 domine A^2 sous la structure D.
>
> Une stratégie est dite non dominée sous D si elle n'est dominée par aucune autre stratégie sous D.

Pour une étude plus rigoureuse des structures de domination, on se référera aux travaux originaux de Yu [Yu73]. Ainsi que nous allons le voir à travers quelques exemples, chaque structure de domination particulière dépend de la quantité et de la

précision des informations dont le décideur dispose pour spécifier ses préférences.

4.4.2. Exemples de structures de domination.

Supposons d'abord que nous puissions quantifier avec exactitude les deux coûts élémentaires suivants :

σ = coût de stockage d'une page en mémoire par unité de temps,

τ = coût du chargement d'une page en mémoire.

Dans ce cas, les stratégies d'allocation peuvent être comparées sur la base d'un seul critère : une stratégie A^1 sera préférée à une autre A^2 si et seulement si :

$$\tau\bar{F}^1 + \sigma\bar{G}^1 \prec \tau\bar{F}^2 + \sigma\bar{G}^2.$$

En d'autres termes, la structure de domination correspondant à cette situation est le demi plan :

$$D(\sigma, \tau) = \{(df, dg) \mid \tau.df + \sigma.dg \prec 0\}. \qquad (4.32)$$

Sous cette structure, les stratégies non dominées sont tout simplement celles minimisant le coût moyen global $\tau\bar{F} + \sigma\bar{G}$.

Considérons maintenant la situation opposée où l'analyste de système est incapable de quantifier l'importance relative à accorder à chacun des coûts F et G. De la sorte, si une stratégie fait mieux qu'une autre stratégie pour ce qui concerne l'un de ces deux coûts, mais fait moins bien en ce qui concerne l'autre coût, l'analyste est également incapable de dire laquelle de ces deux stratégies est "meilleure" que l'autre. Il ne pourra préférer une stratégie A^1 à une autre A^2 que lorsque

$$\bar{F}^1 \prec \bar{F}^2 \quad \text{et} \quad \bar{G}^1 \preceq \bar{G}^2, \quad \text{ou} \quad \bar{G}^1 \prec \bar{G}^2 \quad \text{et} \quad \bar{F}^1 \preceq \bar{F}^2.$$

La structure de domination correspondant à cette situation est donc le quadrant :

$$D_0 = \{(df, dg) \nmid df \leq 0 \text{ et } dg \leq 0\} - (0, 0). \qquad (4.33)$$

Cette structure est généralement connue sous le nom de structure de Pareto [Par96]. Par définition, étant donné une stratégie non dominée sous D_0, il n'existe pas de stratégie générant des coûts $\bar{F}$ et $\bar{G}$ simultanément moindres que cette stratégie non dominée. Autrement dit, parmi toutes les stratégies conduisant à un coût $\bar{F}$ (resp. $\bar{G}$) au plus égal à celui d'une stratégie non dominée sous D_0, cette dernière minimise $\bar{G}$ (resp. $\bar{F}$).

Entre ces deux cas extrêmes, on peut facilement concevoir diverses structures intermédiaires. Ainsi, le concepteur de système peut regarder la mémoire centrale comme une "disponibilité spatiale" qu'il "investit" à des programmes pour obtenir un certain "profit temporel" en retour. Supposons alors que, lorsqu'il compare deux stratégies A^1 et A^2, telles que A^1 requiert davantage d'investissement spatial que A^2 ($\bar{G}^1 - \bar{G}^2 \succ 0$), notre investisseur préférera quand-même $\bar{A}^1$ à $\bar{A}^2$ pourvu que l'amélioration temporelle $\bar{F}^2 - \bar{F}^1$ excède une certaine fraction σ du déficit spatial $\bar{G}^1 - \bar{G}^2$. Ce point de vue conduit à des structures de domination de la forme :

$$D_1(\sigma) = D_0 + \{(df, dg) \nmid dg \succ 0 \text{ et } -df \succ \sigma.dg\}, \ \sigma \succ 0. \qquad (4.34)$$

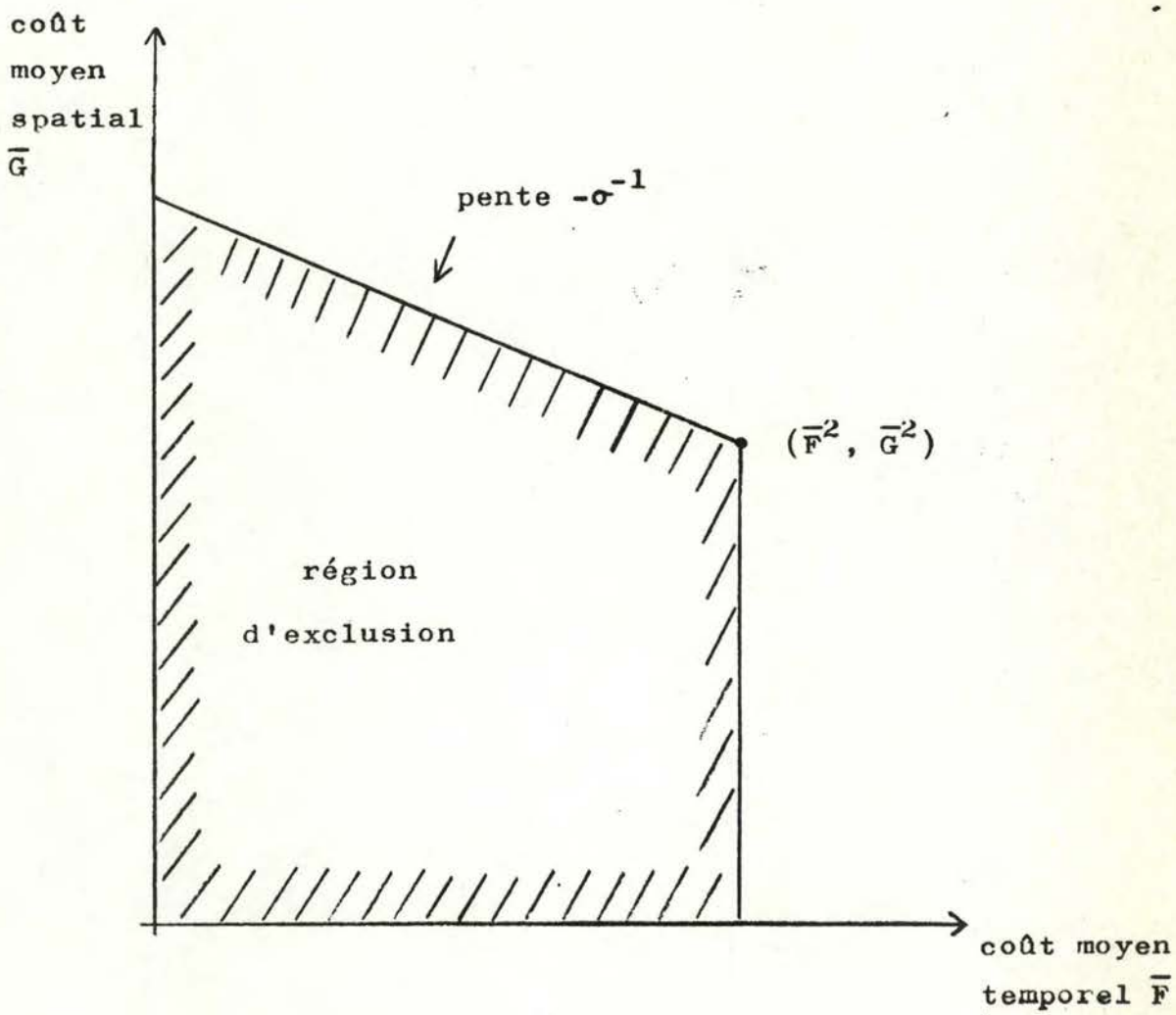
En intervertissant "espace" et "temps" dans le raisonnement heuristique précédent, nous obtenons les structures :

$$D_2(\tau) = D_0 + \{(df, dg) \nmid df \succ 0 \text{ et } -dg \succ \tau.df\}, \ \tau \succ 0. \qquad (4.35)$$

Une illustration géométrique du concept de non domination peut être donnée. Considérons une politique $\bar{A}^2$ qui est non dominée sous une certaine structure D. Par définition, il n'existe aucune politique A^1 pour laquelle $d_{12} \in D$. En d'autres termes, aucune polique ne peut générer un "point" situé dans la région

$$\{(\bar{F}^2+df,\ \bar{G}^2+dg) \mid (df,\ dg) \in D\}.$$

Par exemple, dans le cas de la structure $D_1(\sigma)$, cette région d'exclusion est présentée à la figure ci-dessous.



4.5. Stratégies à l'ensemble d'activité potentiel.

Nous formulons dans ce paragraphe un principe d'allocation conduisant à des stratégies non dominées : les stratégies à l'ensemble d'activité potentiel (ou probabiliste). A tout instant, à part la page actuellement référencée, seules celles dont la probabilité d'être prochainement référencée excède un certain seuil θ sont résidentes en mémoire. Il est évident que chaque modèle de comportement peut conduire à une stratégie à l'ensemble d'activité potentiel particulière. Ainsi, nous verrons que, sous un modèle à références indépendantes, une partie fixe de l'ensemble des pages réside en permanence en mémoire. Le modèle à liste lru conduit tout simplement aux algorithmes LRU à partition fixe, tandis qu'on retrouve les stratégies à l'ensemble d'activité temporel sous certains modèles de lissage exponentiel. Nous établissons enfin la non domination de ces stratégies à l'ensemble d'activité potentiel, sous les différentes structures de domination du § 4.2.2, et pour tout comportement de programme satisfaisant à la CEL (définition 2.6).

4.5.1. Principes de l'ensemble d'activité potentiel.

Définition 4.3.

> Soient S la chaîne de références d'un programme, et p un modèle explicite de prévision de son comportement. A tout instant t, son ensemble d'activité potentiel de niveau θ, θ étant un paramètre positif, est constitué de la page actuellement référencée plus toutes celles dont la prévision

à l'instant prochain est supérieure à θ :

$$PWS_t(\theta) = R_t \cup \{x \mid p_{t+1}(x, S_t) > \theta\}. \quad (4.36)$$

Une stratégie d'allocation A est une stratégie à l'ensemble d'activité potentiel de niveau θ (et déduite du modèle p) si, pour tout t :

$$A_t = PWS_t(\theta). \quad (4.37)$$

Cette stratégie sera notée $PWS(\theta)$.

Ces stratégies sont conceptuellement similaires aux stratégies à l'ensemble d'activité temporel, et la plupart des remarques du § 4.3.1 peuvent être transposées ici, mutatis mutandis. Signalons toutefois que

1) Les stratégies à l'ensemble d'activité potentiel sont définies à partir d'un modèle explicite de comportement de programme, et peuvent s'appliquer à n'importe quel modèle de comportement. Par conséquent, on peut qualifier la présente formulation d'ouverte, en ce sens qu'elle présente un principe pour déduire des règles d'allocation à partir d'une certaine connaissance du comportement de programme, plutôt qu'une certaine règle bien spécifique.

2) Le critère selon lequel les pages résidentes sont déterminées peut, ici aussi, être qualifié de critère à terme immédiat. En effet, à chaque instant, à part la page actuellement requise, seules celles assez susceptibles d'être référencées à l'étape suivantes sont résidentes. Comme dans le contexte d'une allocation à partition fixe, on peut espérer que, pour certains comportements localisés, ce critère est quand-même assez efficace, en dépit de sa simplicité conceptuelle.

3) Pour certains comportements de programme, les stratégies à l'ensemble d'activité potentiel peuvent ne pas être à la demande, au sens de la définition 4.1. Des chargements anticipatifs peuvent avoir lieu au début de chaque nouvelle phase de localisation où les ensembles de pages momentanément privilégiées subissent de brusques renouvellements (voir § 2.3.1). Cependant, si on modélise le comportement à l'intérieur d'une phase par un modèle satisfaisant à la condition explicite de localisation (déf. 2.5), alors les chargements sont à la demande. D'une façon plus précise:

Proposition 4.1.

> Si le modèle de prévision p satisfait à la CEL, alors toute stratégie à l'ensemble d'activité potentiel est à la demande :
>
> $$PWS_{t+1}(\theta) - PWS_t(\theta) \subset R_{t+1}, \quad \forall\ t \text{ et } \theta. \qquad (4.38)$$

Cette proposition se démontre trivialement. En effet, d'après la définition 4.3, pour toute page x appartenant à $PWS_{t+1}(\theta) - PWS_t(\theta)$, (i) ou bien $x = R_{t+1}$, (ii) ou bien $x \neq R_{t+1}$ et $p_{t+1}(x, S_t) \leq \theta < p_{t+2}(x, S_t.R_{t+1})$. Le cas (ii) est impossible, en vertu de la CEL.

Nous nous limiterons essentiellement à des modèles satisfaisant à cette condition, pour lesquels donc les stratégies à l'ensemble d'activité potentiel sont des stratégies à la demande. Dans ce cas, signalons leur analogie avec les stratégies générales à l'ensemble d'activité formulées par Denning et Slutz /DeS76/. On associe à chaque page x une fonction de rétention $Q_t(x)$ qui est remise à zéro après chaque référence, et qui est non décroissante en fonction du temps jusqu'à la prochaine référence à x. L'ensemble d'activité général, à un niveau θ', est constitué, à chaque instant t, de toutes les pages x pour lesquelles $Q_t(x) \leq \theta'$.

4.5.2. Particularisation à quelques modèles de comportement.

1) Le modèle à références indépendantes.

Rappelons que sous ce modèle, la prévision de référence de chacune des pages reste toujours constante, indépendamment du passé. En d'autres termes, quels que soient t, S_t, et x :

$$p_{t+1}(x, S_t) = a(x),$$

où $\{a(x);\ x \in X\}$ est un certain vecteur de prévision donné.

Il est trivial que les pages de l'ensemble suivant :

$$A(\theta) = \{x \mid a(x) \geq \theta\}$$

résident en permanence sous une stratégie à l'ensemble d'activité potentiel de niveau θ.

De plus, si le modèle est vrai pour la chaîne considérée, alors, à tout instant t,

- ou bien $R_t \in A(\theta)$, événement de probabilité $\sum_{x \in A(\theta)} a(x)$, auquel cas nous avons

$$PWS_t(\theta) = A(\theta) \text{ et } |PWS_t(\theta)| = |A(\theta)|,$$

- ou bien $R_t = y$, $y \notin A(\theta)$, événements de probabilités respectives $a(y)$, auquel cas

$$PWS_t(\theta) = y + A(\theta) \text{ et } |PWS_t(\theta)| = |A(\theta)| + 1.$$

La stratégie $PWS(\theta)$ est donc une stratégie à partition variable, sous laquelle une partie fixe $A(\theta)$ de l'ensemble des pages réside en permanence en mémoire centrale, les autres pages étant libérées aussitôt qu'elles ne sont plus requises par le processeur.

2) Le modèle à liste lru.

Rappelons que sous ce modèle, la prévision de référence de chacune des pages est fonction uniquement de la position actuelle de la page dans la pile lru. Quels que soient t, S_t, et x :

$$p_{t+1}(x, S_t) = b_i,$$

où $i = m_{t+1}(x, S_t)$ est le rang de la page x dans la pile lru (voir § 2.3.3), et où $\{b_i ; i = 1, \ldots, n\}$ est un vecteur de probabilité fixé.

Considérons l'ensemble des rangs des pages résidentes sous une stratégie à l'ensemble d'activité potentiel de niveau θ :

$$\begin{aligned}\{m_{t+1}(x, S_t) \nmid x \in PWS_t(\theta)\} &= \{m_{t+1}(R_t, S_t)\} \cup \{m_{t+1}(x, S_t) \nmid b_{m_{t+1}(x, S_t)} > \theta\} \\ &= \{1\} \cup \{i \nmid b_i > \theta\},\end{aligned}$$

indépendamment de t et de S_t.

Cet ensemble étant fixe, il en résulte notamment que PWS(θ) est une stratégie à partition fixe opérant avec

$$c(\theta) = |\{1\} \cup \{i \nmid b_i > \theta\}|$$

cadres.

Supposons de plus que les b_i sont monotones non croissants avec i. Dans ce cas, le modèle satisfait à la CEL; de plus :

$$c(\theta) = \begin{cases} 1, & \text{si } b_1 \leq \theta \\ \max\{i \nmid b_i > \theta\}, & \text{dans le cas contraire} \end{cases},$$

et PWS(θ) n'est alors autre que l'algorithme LRU qui opère avec $c(\theta)$ cadres.

3) Le modèle de lissage exponentiel.

Considérons le cas d'un modèle homogène pour lequel, quels que soient x, t, et S_t :

$$p_{t+1}(x, S_t) = (1-\alpha)p_t(x, S_{t-1}) + \begin{cases} \alpha & \text{si } x = R_t \\ 0 & \text{si } x \neq R_t \end{cases},$$

le paramètre de lissage α étant un réel compris entre 0 et 1. Nous allons montrer les propriétés suivantes, en relation avec les stratégies à l'ensemble d'activité temporel. Pour tout t :

$TWS_t(h) \subset PWS_t(\theta)$, pour $\theta < \alpha(1-\alpha)^{h-1}$, et

$TWS_t(h) \supset PWS_t(\theta)$, pour $\theta \geq (1-\alpha)^h$.

Ces inclusions impliquent notamment l'identité $PWS_t(\theta) = TWS_t(h)$ pour $(1-\alpha)^h \leq \theta < \alpha(1-\alpha)^{h-1}$ (cela n'est possible que si $\alpha > 1/2$).

En effet, à tout instant t, désignons l'instant de la dernière référence à toute page x par :

$$t_x = \max \{u \mid u \leq t \text{ et } R_u = x\}.$$

Nous avons, à partir de l'expression récurrentielle qui définit le modèle :

$$p_{t+1}(x, S_t) = (1-\alpha)^{t-t_x} p_{t_x+1}(x, S_{t_x})$$

$$= (1-\alpha)^{t-t_x} ((1-\alpha)p_{t_x}(x, S_{t_x-1}) + \alpha).$$

Les inclusions à démontrer sont alors immédiates puisque :

- pour $x \in TWS_t(h)$: $t-t_x \leq h-1$, de sorte que
$$p_{t+1}(x, S_t) \geq (1-\alpha)^{h-1} \alpha;$$
- pour $x \notin TWS_t(h)$: $t-t_x \geq h$, et
$$p_{t+1}(x, S_t) \leq (1-\alpha)^h .$$

4.5.3. Non domination de ces stratégies.

Nous considérons dans ce paragraphe un programme obéissant à un modèle explicite de comportement qui satisfait à la CEL. Sous les diverses structures de domination envisagées au § 4.2.2, nous montrons alors qu'il est chaque fois possible de choisir le niveau θ pour que la stratégie à l'ensemble d'activité potentiel correspondante soit non dominée.

Dans ce qui va suivre, A^2 désigne une stratégie à l'ensem- d'activité potentiel à un certain niveau θ, et A^1 une stratégie quelconque. Comme au § 4.4, les quantités définies pour chacune de ces deux stratégies seront indicées par "1" ou "2" selon qu'elles se rapportent à A^1 ou à A^2. Ainsi :

$$f_t^i = f_t(A^i), \text{ et } g_t^i = g_t(A^i), \quad \forall\, i = 1, 2 \text{ et } t \geq 0.$$

Nous surmonterons également une variable d'une barre pour représenter son espérance mathématique.

Avec ces conventions d'écriture, nous avons alors le

Théorème 4.4.

> Soit une chaîne de références obéissant à un modèle explicite p qui satisfait à la CEL. Alors, pour toute stratégie A^1 et pour tout niveau θ, nous avons :
>
> $$\bar{F}^1 - \bar{F}^2 \geq -\theta(\bar{G}^1 - \bar{G}^2). \tag{4.39}$$

Démonstration.

Il suffit de démontrer que :

$$\bar{f}_t^1 - \bar{f}_t^2 \geq -\theta(\bar{g}_t^1 - \bar{g}_t^2), \quad \forall\, t \geq 0. \tag{4.40}$$

Notons que, en vertu de la Prop. 4.1 (p. 20), A^2 (= PWS(θ)) est à la demande, de sorte que :

$$f_t^2 = \begin{cases} 1, \text{ si } R_{t+1} \notin A_t^2, \\ 0, \text{ dans le cas contraire.} \end{cases}$$

A^1 étant quelconque, la contrainte d'allocation (4.2) implique :

$$f_t^1 \geq \begin{cases} 1, \text{ si } R_{t+1} \notin A_t^1 \\ 0, \text{ dans le cas contraire.} \end{cases}$$

Cela étant, conditionnellement à des réalisations spécifiques s_t, a^1 et a^2 de S_t, A_t^1 et A_t^2 respectivement, nous avons :

$$\bar{f}_t^2(s_t) \overset{\text{déf}}{=} E[f_t^2 \mid S_t=s_t] = Pr[R_{t+1} \notin a^2 \mid S_t=s_t],$$

$$\bar{f}_t^1(s_t) \overset{\text{déf}}{=} E[f_t^1 \mid S_t=s_t] \geq Pr[R_{t+1} \notin a^1 \mid S_t=s_t].$$

D'où :

$$\bar{f}_t^1(s_t) - \bar{f}_t^2(s_t) \geq Pr[R_{t+1} \notin a^1 \mid S_t=s_t] - Pr[R_{t+1} \notin a^2 \mid S_t=s_t]$$

$$= \sum_{x \in a^2-a^1} p_{t+1}(x, s_t) - \sum_{x \in a^1-a^2} p_{t+1}(x, s_t).$$

Or, en vertu de la définition de l'ensemble d'activité potentiel :

(i) pour $x \in a^2-a^1 = PWS_t(\theta)-A_t^1$: $p_{t+1}(x, s_t) > \theta$, et

(ii) pour $x \in a^1-a^2 = A_t^1-PWS_t(\theta)$: $p_{t+1}(x, s_t) \leq \theta$.

Par conséquent :

$$\bar{f}_t^1(s_t) - \bar{f}_t^2(s_t) \geq \theta(|a^2-a^1| - |a^1-a^2|)$$

$$= -\theta(|a^1| - |a^2|),$$

d'où l'inégalité (4.40). C.Q.F.D.

Le théorème précédent peut encore s'énoncer comme suit. Si A^2 désigne une stratégie d'allocation à l'ensemble d'activité potentiel à un niveau θ, alors, pour toute stratégie A^1 :

$$(\bar{F}^1 - \bar{F}^2) + \theta(\bar{G}^1 - \bar{G}^2) \geq 0.$$

Autrement dit, nous avons toujours :

$$d_{12} \notin D_\theta,$$

où :

$$D_\theta = \{(df, dg) \mid df + \theta.dg < 0\}. \qquad (4.41)$$

Un premier corollaire du théorème 4.3 est donc :

Corollaire 1.

> Sous les conditions du théorème 4.3, la stratégie à l'ensemble d'activité potentiel au niveau $\theta = \sigma/c$ minimise le coût moyen global $c\bar{F} + \sigma\bar{G}$.

En ce qui concerne maintenant la structure de Pareto, notons l'inclusion :

$$D_0 \subset D_\theta, \ \forall \ \theta > 0. \qquad (4.42)$$

Le théorème implique donc, a fortiori, que :

$$d_{12} \notin D_0,$$

pour toute stratégie A^1. En d'autres termes, nous avons le

Corollaire 2.

> Sous les conditions du théorème 4.3, toute stratégie à l'ensemble d'activité potentiel est non dominée sous D_0.

Le théorème précédent se particularise tout aussi facilement aux structures $D_1(\sigma)$ et $D_2(c)$ du § 4.4.2. Ainsi, nous avons :

$$D_1(\sigma) \subset D_\theta, \text{ pour } c \leq \sigma. \qquad (4.43)$$

(En effet, en vertu de (4.34), si $(df, dg) \in D_1(\sigma)$, alors :

ou bien (i) $(df, dg) \in D_0$,

ou bien (ii) $-df \succ \sigma.dg$ et $dg \succ 0$.

Dans le cas (i) : $(df, dg) \in D_\theta$, en vertu de (4.42).

Dans le cas (ii), nous avons, pour $\theta \leq \sigma$:

$df + \theta.dg \leq df + \sigma.dg \prec 0$,

et $(df, dg) \in D_\theta$ également).

D'où le

Corollaire 3.

> Sous les conditions du théorème 4.3, la stratégie PWS(θ) est non dominée sous les structures $D_1(\sigma)$, pourvu que $\theta \leq \sigma$.

D'une façon similaire, l'inclusion :

$$D_2(\tau) \subset D_\theta, \text{ pour } \theta \geq 1/\tau \qquad (4.44)$$

conduit au

Corollaire 4.

> Sous les conditions du théorème 4.3, la stratégie PWS(θ) est non dominée sous les structures $D_2(\tau)$, pourvu que $\theta \geq 1/\tau$.

Ce théorème nous permet également de donner une réfutation constructive à la conjecture mentionnée à la fin du § 4.3.2. Considérons en effet une chaîne de références obéissant à un modèle à liste lru dont les probabilités de référence b_i sont non croissantes. Ce modèle satisfait à la CEL, et $\delta(h)$ est convexe, en vertu du théorème 4.3. D'autre part, nous avons vu à l'exemple 2 du § 4.3.2 que les stratégies à l'ensemble d'activité potentiel sont alors les stratégies LRU à partition fixe, qui sont alors non dominées.

4.6. Conclusions.

Intuitivement, on peut justifier l'allocation à partition variable de la façon suivante. Les besoins en espace d'un programme sont loin d'être constants au cours du temps. Au contraire, il existe des phases où les références sont plus dispersées que dans d'autres, et vice versa. Dès lors, plutôt que d'allouer à chaque programme une zone d'espace fixe, il vaudrait mieux permettre une utilisation moins rigide de la mémoire centrale, où chacun des programmes peut acquérir ou céder de l'espace selon ses besoins propres et instantanés.

Les premières stratégies à partition variable étudiées expérimentalement sont probablement les stratégies de remplacement biaisées [BeI67]. On favorise alternativement chacun des programmes en lui allouant davantage de cadres de mémoire pendant certaines périodes choisies tout à fait arbitrairement. Les bonnes performances observées pour ces stratégies peuvent s'expliquer par certaines propriétés de convexité du taux de défauts de page en fonction de l'espace alloué [BeK69, DeS73].

Ce genre de stratégies ne reflète pas un principe fondamental de toute allocation à partition variable, qui doit être, à notre avis, une allocation en fonction des besoins des programmes. Une des formulations les plus connues de ce principe est peut être celui de l'ensemble d'activité temporel de Denning : à tout moment de l'exécution d'un programme, les pages "nécessaires", qui doivent donc résider en mémoire centrale, sont considérées comme celles effectivement requises durant

un passé récent, en conformité avec la localisation des références /Den68a, Den68b/. Une autre stratégie à partition variable est la stratégie PFF de Chu et Opderbeck /Ch072, OCh74/. Ce genre de stratégies se base sur la fréquence des défauts de page passés de chacun des programmes pour estimer leurs besoins actuels en espace. Ainsi, les programmes ayant subi de fréquents défauts de page peuvent acquérir des cadres supplémentaires au détriment des autres programmes.

Comparativement à ces stratégies, les stratégies à l'ensemble d'activité potentiel formulées dans ce chapitre représentent davantage un "principe" d'allocation plutôt que des règles opératoires bien déterminées. Elles suggèrent comment le principe de l'offre en fonction de la demande devrait être transposé dans le contexte de la gestion d'une mémoire virtuelle. A cause de leur simplicité conceptuelle, la formulation même de ces stratégies peut ressembler à une lapalissade : à tout moment, seules doivent être en mémoire centrale les pages suffisamment probables d'être prochainement référencées. De la sorte, on vise, à tout instant

- à remplir la mémoire centrale d'informations les plus utiles actuellement, en prévention des demandes futures;
- à permettre d'utiliser toute cette mémoire comme un pot commun dans lequel chaque programme peut puiser selon ses besoins du moment.

L'approche choisie pour analyser les stratégies d'allocation est une approche marginale. L'allocation d'espace est

analysée séparément pour chaque programme, et, pour chaque programme, séparément pour chaque instruction élémentaire. De plus, l'allocation de la mémoire est dissociée des autres fonctions du système d'exploitation, telles que l'ordonnancement des programmes au niveau du processeur ou au niveau de la mémoire auxiliaire de pagination. De la sorte, une critique majeure qu'on peut faire à propos de cette approche est qu'elle ignore les aspects ayant trait à la coordination temporelle de ces diverses fonctions.

Les interactions entre ces fonctions, les effets de la concurrence entre programmes sur les performances du système ont été modélisés par des réseaux d'attente plus ou moins complexes [Buz71, Cou71, Gel73b, Bra74, BCMP74, ReK75, TQT76a, ...]. La résolution de ce genre de modèles devrait permettre une meilleure compréhension des principes de contrôle optimal des systèmes informatiques [BGLP75, LeP76, DeK76, ...].

4.7. Annexe : démonstration du théorème 4.1 (p. 6).

Démonstration du théorème 4.1.

A partir de la stratégie d'allocation $A^1 = A_0^1.A_1^1...A_t^1...$, nous définirons A^2 de la façon suivante :

$$A_0^2 = A_0^1, \tag{4.45}$$

et, d'une manière récursive :

$$A_{t+1}^2 = A_t^2 \cup R_{t+1} - Z_t^2, \tag{4.46}$$

où :

$$Z_t^2 = A_t^2 - A_{t+1}^1. \tag{4.47}$$

Il est évident que la stratégie A^2 ainsi définie est une stratégie à la demande. Notons également que si A^1 est non prospective, c'est-à-dire si chacun des états A_t^1 ne dépend que du passé S_t, alors il en sera de même pour A^2.

Nous poserons, pour alléger les notations :

$$F_t^i = F_t(A^i),\ G_t^i = G_t(A^i),\ i = 1, 2. \tag{4.48}$$

Nous allons démontrer les inégalités suivantes, qui impliquent les inégalités (4.10) et (4.11) :

$$F_t^2 \leq F_t^1 - |A_t^1 - A_t^2|,\ \forall\ t \geq 0; \tag{4.49}$$

$$G_t^2 \leq G_t^1 - |A_{t-1}^1 - A_{t-1}^2|,\ \forall\ t = 0. \tag{4.50}$$

Démonstration de (4.49).

Cette inégalité est triviale pour $t = 0$ ($F_0^1 = F_0^2 = 0$). Supposons-la vraie pour $t \geq 0$. Nous avons alors :

$$F^2_{t+1} = F^2_t + |A^2_{t+1} - A^2_t|, \text{ en vertu de (4.3) et (4.4)}$$
$$\leq F^1_t - |A^1_t - A^2_t| + |A^2_{t+1} - A^2_t|, \text{ par induction}$$
$$= F^1_{t+1} - |A^1_{t+1} - A^1_t| - |A^1_t - A^2_t| + |A^2_{t+1} - A^2_t|$$
$$\leq F^1_{t+1} - |A^1_{t+1} - A^2_t| + |A^2_{t+1} - A^2_t|.$$

Au lemme 4.3, nous démontrerons l'identité suivante :

$$A^1_{t+1} - A^2_t = (A^1_{t+1} - A^2_{t+1}) + (A^2_{t+1} - A^2_t), \quad \forall\, t \geq 0$$

("+" est mis pour la réunion d'ensembles disjoints). Il en résulte alors l'identité entre les cardinaux :

$$|A^1_{t+1} - A^2_{t+1}| = |A^1_{t+1} - A^2_t| - |A^2_{t+1} - A^2_t|,$$

ce qui termine la démonstration par induction de (4.49).

Démonstration de (4.50).

Cette inégalité est triviale pour $t = 0$ ($G^1_0 = G^2_0 = 0$). Supposons-la vraie pour $t \geq 0$. Nous avons alors :

$$G^2_{t+1} = G^2_t + |A^2_t|, \text{ en vertu de (4.5) et (4.6)}$$
$$\leq G^1_t + |A^2_t|, \text{ par induction}$$
$$= G^1_{t+1} - |A^1_t| + |A^2_t|.$$

Le lemme 4.2 qui va suivre établira l'inclusion suivante :

$$A^2_t \subset A^1_t, \quad \forall\, t \geq 0.$$

Il en résulte alors :

$$|A^1_t - A^2_t| = |A^1_t| - |A^2_t|,$$

ce qui termine la démonstration par induction de (4.50).

C.Q.F.D.

Lemme 4.2.

Si A^2 est définie à partir de A^1 par les équations (4.45-47), alors :

$$A^2_t \subset A^1_t, \quad \forall\, t \geq 0.$$

Démonstration.

Cette inclusion est triviale pour $t = 0$. Pour tout $t \geq 0$, nous avons :

$$\begin{aligned} A^2_{t+1} - A^1_{t+1} &= (A^2_t \cup R_{t+1} - Z^2_t) - A^1_{t+1} \\ &= (A^2_t - Z^2_t) - A^1_{t+1}, \text{ car } R_{t+1} \in A^1_{t+1} \\ &= A^2_t - (Z^2_t \cup A^1_{t+1}) \\ &= A^2_t - A^2_t, \text{ en vertu de (4.47)} \\ &= \emptyset. \quad \text{C.Q.F.D.} \end{aligned}$$

Lemme 4.3.

Si A^2 est définie à partir de A^1 par les équations (4.45-47), alors :

$$A^1_{t+1} - A^2_t = (A^1_{t+1} - A^2_{t+1}) + (A^2_{t+1} - A^2_t), \quad \forall\, t \geq 0.$$

Démonstration.

Comme $A^2_{t+1} \subset A^1_{t+1}$ en vertu du lemme 4.2 :

$$A^1_{t+1} - A^2_t = ((A^1_{t+1} - A^2_{t+1}) - A^2_t) + (A^2_{t+1} - A^2_t).$$

Il suffit alors de démontrer que :

$$(A^1_{t+1} - A^2_{t+1}) - A^2_t = A^1_{t+1} - A^2_{t+1}$$

c'est-à-dire que A^2_t est disjoint de $A^1_{t+1} - A^2_{t+1}$. Nous avons :

$$\begin{aligned} A^2_t \cap (A^1_{t+1} - A^2_{t+1}) &= (A^2_t - A^2_{t+1}) \cap A^1_{t+1} \\ &= Z^2_t \cap A^1_{t+1}, \text{ en vertu de (4.46)} \\ &= \emptyset, \text{ en vertu de (4.47). C.Q.F.D.} \end{aligned}$$

5. ASPECTS DU CONTROLE GLOBAL DES SYSTEMES A MEMOIRE VIRTUELLE

5.1. Présentation du chapitre.

Aux chapitres précédents, nous avons étudié le processus de pagination d'un programme avec comme objectifs des critères de performance propres au programme considéré. Ainsi, sous une allocation à partition fixe, nous avons recherché les principes des stratégies de remplacement minimisant le taux de défauts de page, et, sous une allocation à partition variable, nous avons introduit un second critère, la minimisation de l'espace alloué au programme. Nous nous proposons maintenant d'analyser les interactions de ces performances marginales sur l'efficacité globale du système, mesurée par le taux d'utilisation de certaines unités "essentielles" du système, le processeur central par exemple. Nous présenterons également diverses règles de contrôle optimal du système.

Un nouveau programme qui arrive réside d'abord sur une file d'attente externe jusqu'à ce qu'il soit admis dans le système. Le système est constitué d'une mémoire centrale (MC) de taille fixée, et d'un réseau d'unités de service. Ce réseau comprend un processeur central (PC), un disque de pagination (DP), prolongement de la MC, et un certain nombre d'unités d'entrée-sortie (ES). Pendant son exécution, chaque programme boucle dans le réseau de la façon suivante : il nécessite une certaine durée de traitement au PC jusqu'à ce qu'il subit un défaut de page ou requiert une entrée-sortie ; il reviendra

alors au PC lorsque la page manquante est chargée ou l'entrée sortie effectuée (Fig. 1).

Une approche fréquente pour modéliser la concurrence entre les programmes au niveau de la MC suppose un équipartage tel que chacun des programme a le même taux de défauts de page ; évidemment, ce taux augmente avec le nombre de programmes dans le système, puisque chacun d'eux dispose de moins d'espace.

En résumé, le modèle étudié est un réseau ouvert de files d'attente dans lequel les caractéristiques opérationnelles des programmes (notamment leur taux de défauts de page) varient avec le nombre total de programmes dans la mémoire.

A cause de cette dépendance, le réseau considéré n'est pas un réseau de Jackson [Jac63] où la durée de service à chaque station ne peut dépendre que de la longueur de la file à cette station. Néanmoins, son analyse peut se ramener à celle de réseaux de Jackson par l'intermédiaire d'une méthode de décomposition due à Courtois [Cou71].

D'une façon informelle, et dans le contexte du problème particulier qui nous intéresse, la décomposition du problème peut se justifier par le fait que les événements (changements d'état du système) se décomposent naturellement en deux classes suivant leurs fréquences relatives :

- d'une part, ceux provoquant un changement du nombre n de programmes dans le système, c'est-à-dire les entrées de nouveaux programmes ou les départs de programmes arrivés à leur fin ; ces événements sont relativement espacés, un ordre de grandeur

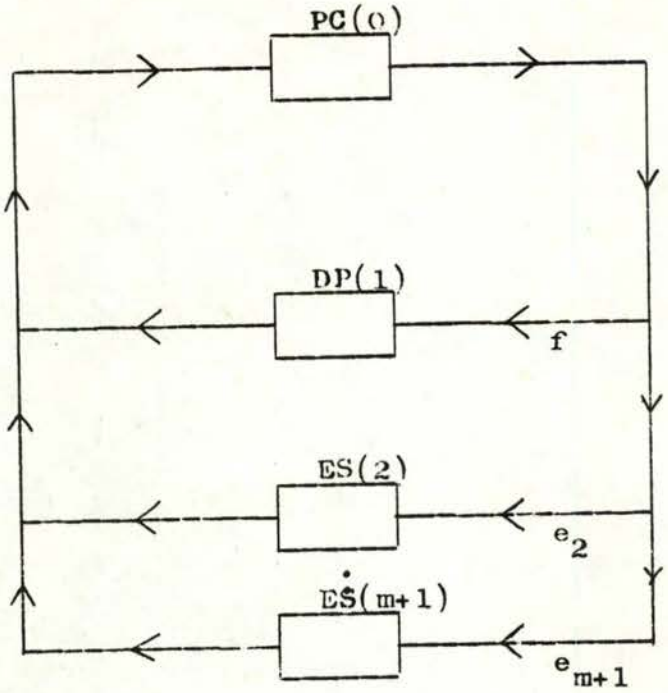


Fig. 1. Configuration du système.

de leur espacement étant généralement la minute ;
- d'autre part, les transitions des programmes à l'intérieur du système, c'est-à-dire les passages des programmes d'une file interne à une autre ; ces événements, consécutifs à des fins de traitements élémentaires à une unité du système (PC, DP, ou ES) se passent généralement à l'échelle de la seconde, voire de la milliseconde.

A cause de cette disproportion entre ces deux types de phénomènes, le comportement interne du système pendant une période où le nombre de programmes dans le système est constant, mettons n, peut être approximé par le comportement en régime permanent d'un système fermé de n programmes. L'analyse des performances du système sous une charge fixe n a été effectuée par de nombreux auteurs, et les principaux résultats seront rappelés au § 5.2. On observe généralement que, lorsque le degré de multiprogrammation n augmente, les taux d'utilisation du PC et des ES commencent d'abord par croître, puis atteignent leur maximum pour une certaine valeur n_0 de n, et enfin se détériorent très rapidement. Bien que l'évaluation numérique du degré de multiprogrammation optimal n_0 ne présente généralement aucune difficulté majeure, son évaluation analytique n'a pu être obtenue que dans certains cas particuliers [BGLP73].

Du point de vue pratique, le problème de la caractérisation d'une charge optimale consiste à rechercher des critères "mesurables" permettant d'inférer si un système est actuellement surchargé ou sous utilisé. Parmi de telles approches, signalons

notamment celle de Leroudier et Potier préconisant une régulation adaptative de la charge basée sur le taux d'utilisation du DP (/LeP76/, règle des 50 %), et celle de Denning et Kahn, basée sur la durée moyenne de temps virtuel entre 2 défauts de page (/DeK76/, règle L=S) : une forte utilisation de l'unité de pagination, ou des défauts de page trop fréquents peuvent constituer des critères indiquant une surcharge du système, et impliquent la désactivation de certains programmes du système.

L'analyse effectuée au paragraphe 5.2, bien que permettant de dégager certains aspects d'une multiprogrammation optimale, et de mettre en évidence le phénomène de l'écroulement (thrasing) comme conséquence de l'insuffisance de l'espace en MC alloué aux programmes, n'en escamote pas moins certains niveaux décisionnels cruciaux. Notamment, lorsque les programmes ont des caractéristiques opérationnelles différentes, il se pose le problème de l'affectation des disciplines de priorité (scheduling) aux diverses unités de service du système. Dans un réseau fermé, la recherche des règles de priorité maximisant le taux d'utilisation du CP conduit généralement à l'optimisation d'une fonction rationnelle de plusieurs variables ; comme le fait remarquer Spirn /Spi76/, le nombre de ces variables croît proportionnellement avec le nombre d'unités de service du réseau, et exponentiellement avec le nombre de programmes dans le système. A cause de cette extrême complexité, les politiques d'ordonnancement optimales n'ont pu être déterminées que dans

des cas fort simples. Ainsi, Spirn [Spi76] considère le cas de 2 programmes bouclant alternativement entre un PC et un DP. Nous présenterons au § 5.4 un modèle similaire, mais où le DP représente le principal goulot d'étranglement du système. L'analyse de ces modèles permet d'inférer que la règle de priorité optimale au DP est d'accorder la priorité préemptive aux programmes les moins fréquemment sujets à des défauts de page, tandis que la règle de priorité au PC n'a généralement que très peu d'influence sur l'efficacité du système. Nous donnerons également une procédure permettant de partitionner d'une façon optimale une MC de taille donnée entre 2 programmes. Cette procédure conduit généralement à des partitions inégales, ce qui est en accord avec d'autres études antérieures [BeK69, DeS73].

5.2. Le modèle de multiprogrammation classique.

Pour analyser l'effet de la charge sur les performances globales du système, le modèle suivant a été utilisé, à quelques variantes près, par la plupart des auteurs.

Un nombre fixe n de programmes circule dans le réseau fermé de la Fig. 1. Rappelons que ce réseau comprend un PC, encore appelé serveur central [Buz71] ou serveur d'échange [Bra71], un DP, et m unités d'ES. Les n programmes sont supposés avoir des caractéristiques opérationnelles identiques. Ces caractéristiques sont :

$1/\sigma_1$ = durée moyenne d'un transfert de page ;

$1/\sigma_i$ = durée moyenne d'un échange à l'unité d'ES i, $i = 2,\ldots,m+1$;

$1/f$ = durée moyenne de temps virtuel entre 2 défauts de page ;

$1/e_i$ = durée moyenne de temps virtuel entre 2 demandes de l'unité d'ES i.

Désignons par A_0, A_1 et A_i $(i=2,\ldots,m+1)$ les taux d'utilisation en régime permanent du PC, du DP et des unités d'ES respectivement.

5.2.1. Résultats exacts.

Tout d'abord, en utilisant un théorème dû à Chang et Lavenberg [ChL72], on peut montrer les relations suivantes entre ces différents taux d'utilisation

$$fA_0 = \sigma_1 A_1, \tag{5.1}$$

$$e_i A_0 = \sigma_i A_i, \quad i=2,\ldots,m+1. \tag{5.2}$$

Ces relations, généralement appelées lois de conservation [Buz71, Buz76], expriment qu'en régime permanent, le nombre moyen de programmes arrivant à chaque serveur par unité de temps est égal au nombre moyen de programmes quittant ce serveur par unité de temps. Ces lois permettent trivialement le calcul de tous les taux d'utilisation à partir de l'un d'entre eux.

La solution analytique de ce modèle a été obtenue dans le cas où toutes les durées mentionnées au début de cette section sont distribuées exponentiellement. En effet, le modèle est alors un cas particulier d'un réseau de Jackson [Jac63], et les probabilités à l'état stationnaire :

$$p(n_1,n_2,\ldots,n_{m+1}) = \lim_{t\to\infty} \Pr[\text{à l'instant } t, \text{ nb. de prog. au DP} = n_1,\ldots, \text{nb. de prog. à l'ES } i = n_i]$$

$$\left(\sum_{i=1}^{m+1} n_i \leq n\right)$$

valent [Buz71] :

$$p(n_1,n_2,\ldots,n_{m+1}) = \frac{1}{G(n)} (\sigma_1/f)^{n_1} \prod_{i=2}^{m+1} (\sigma_i/e_i)^{n_i} \tag{5.3}$$

où la constante de normalisation $G(n)$ vaut évidemment :

$$G(n) = \sum_{\sum_{i=1}^{m+1} n_i \leq n} (\sigma_1/f)^{n_1} \prod_{i=2}^{m+1} (\sigma_i/e_i)^{n_i}. \tag{5.4}$$

Le taux d'utilisation du PC, A_0, s'obtient alors en sommant les probabilités élémentaires (5.3) sur tous les états pour lesquels le nombre de programmes au PC, $n - \sum_{i=1}^{m+1} n_i$, est positif ; Buzen [Buz71] a montré que ce taux vaut :

$$A_0 = \frac{G(n-1)}{G(n)} , \qquad (5.5)$$

et a présenté dans /Buz73/ une procédure numérique pour le calcul de ce taux.

Dans l'étude de l'influence de la charge n sur A_0, il est indispensable de spécifier comment les caractéristiques des programmes, particulièrement le taux de défauts de page, varient avec ce nombre n. La méthode classique consiste à supposer que chacun des programmes possède la même fonction durée de vie L (lifetime function /BeK69/) ; pour toute valeur particulière c, L(c) représente la durée moyenne de temps virtuel entre 2 défauts de page d'un programme lorsque c cadres de page sont alloués à ce dernier. La mémoire centrale, de taille C cadres, sera de plus supposée équipartagée entre les différents programmes. De la sorte, sous une charge n, le taux de défauts de page d'un programme vaut :

$$f = 1/L(\frac{C}{n}) . \qquad (5.6)$$

Cette fonction L a fait l'objet de nombreuses études expérimentales en raison du rôle central qu'elle joue dans l'analyse des systèmes à mémoire virtuelle, et de certaines règles heuristiques de gestion de cette mémoire (voir § 5.2.2). Typiquement, L est une fonction monotone croissante en c, ayant l'allure de la lettre S (voir Fig.2). Généralement, on observe que L est convexe avant une certaine valeur c_g de c, et concave après.

Ainsi, l'amélioration de la durée de vie, obtenue en augmentant l'espace alloué aux programmes, commence d'abord par être très forte : un programme nécessite un "minimum" d'espace. Néanmoins, au delà de ce minimum, cette amélioration devient de plus en plus négligeable. La valeur critique c_g est appelée, d'une façon assez suggestive, le genou (knee ; [DeG75]).
Graphiquement, c'est le point maximisant le rapport : L(c)/c.

Une approximation de L, dans le domaine où cette fonction est convexe, a été proposée par Belady et Kuehner [BeK69] :

$$L(c) = a.c^k, \quad a > 0, \quad k \geq 1, \tag{5.7}$$

où a est une constante liée à la vitesse du PC, et où k est une mesure de la localisation propre au programme ; généralement, k est compris entre 1.0 et 2.0, et peut même excéder 2.0 pour des programmes fortement localisés.

Notons que l'approximation du taux de défauts de page que nous avons donné lors de l'étude du modèle de lissage (théorème 3.7) conduit également à une approximation convexe de L :

$$L(c) = a.(1-\alpha)^{-c}, \quad a > 0, \quad \alpha \in [0, 1], \tag{5.8}$$

le coefficient α étant aussi une mesure de la localisation.

Chamberlin et al [CFL73] ont proposé l'approximation :

$$L(c) = b / (1 + (d/c)^2), \tag{5.9}$$

où b représente la plus grande valeur possible de L, et d la mémoire nécessaire pour que le programme ait une durée de vie égale à la moitié de sa valeur maximale b. On peut vérifier très facilement que cette valeur d est aussi précisément le genou de la courbe (5.9) : $L(d)/d \geq L(c)/c$, pour tout c.

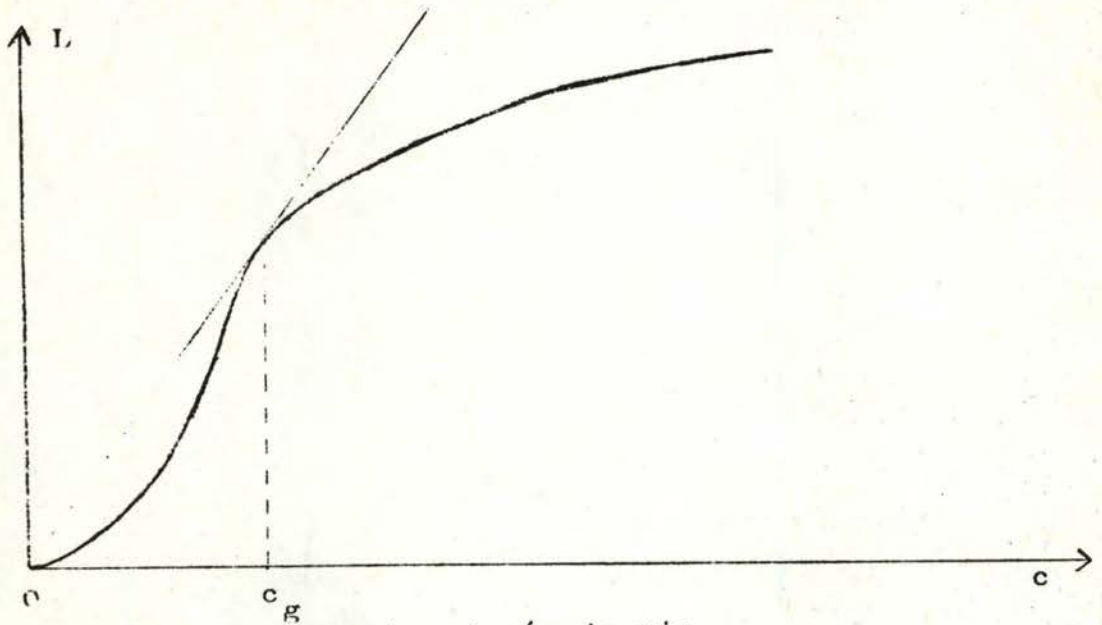


Fig. 2. Fonction durée de vie.

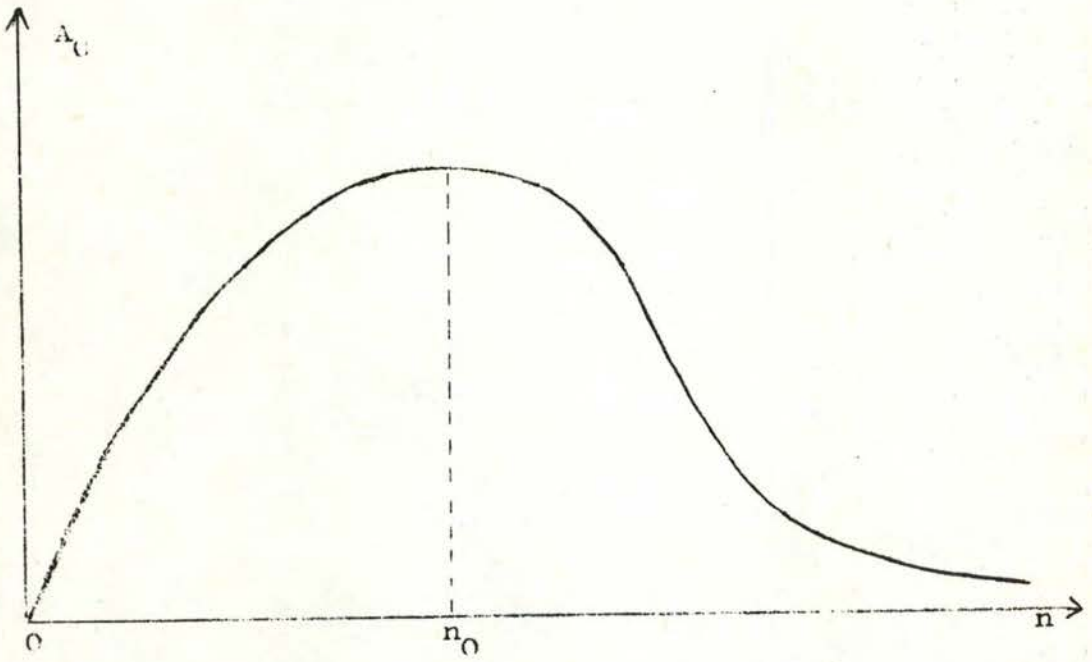


Fig. 3. Taux d'utilisation du processeur central

5.2.2. Conclusions expérimentales.

En se fixant la fonction durée de vie L, ainsi que les paramètres σ_i, e_i, le calcul de A_0, pour diverses valeurs de n, ne présente aucune difficulté particulière et a été effectué par de nombreux auteurs. Sans prétendre à l'exhaustivité, signalons les travaux [Cou71, BGLP73, Bra74, LeP76, DeK76].

Pour L, σ_i, e_i fixés, on constate toujours que la courbe représentant A_0 en fonction du degré de multiprogrammation n a une forme de cloche comme il est indiqué à la Fig. 3. Intuitivement, une telle propriété est tout à fait évidente : l'augmentation de n a deux effets contradictoires sur A_0 : d'une part, en accroissant la possibilité de parallélisme entre les diverses unités du système, on peut améliorer leur taux d'utilisation, en particulier celui du processeur ; mais, d'autre part, plus il y a de programmes dans le système, moins chacun d'eux dispose d'espace en mémoire centrale, d'où des défauts de page plus fréquents, et un goulot d'étranglement au niveau du disque de pagination.

La détermination du degré de multiprogrammation optimal n_0, correspondant à une utilisation maximale du PC, pose deux problème majeurs :

- la maximisation analytique de A_0, à partir des équations (5.4-6), est généralement hors de question, vu la complexité de ces équations ;

- même si cette maximisation est techniquement possible, par tabulation de A_0 pour différentes valeurs de n, cette approche suppose L non seulement connue, mais de surcroit stationnaire dans le temps.

Ces problèmes ont orienté les recherches actuelles vers des critères de contrôle approximatifs certes, mais plus pratiques et plus adaptatatifs que celui basé sur un degré de multiprogrammation théoriquement idéal.

Règle du genou /DeG75, DeK76/.

Un compromis possible entre la maximisation de la durée de vie d'un programme et la minimisation de l'espace qui lui est alloué peut inciter à accorder au programme un nombre c_g de cadres qui maximise le rapport $L(c)/c$. L'utilisation d'une telle méthode d'allocation nécessite évidemment que la fonction L soit estimée avec la plus grande précision. D'un point de vue numérique, cette règle a été justifiée par Denning et Kahn /DeK76/ qui ont constaté, dans tous les cas qu'ils ont étudiés, que le degré de multiprogrammation préconisé par la règle du genou, $n_g = C/c_g$, est assez voisin de la valeur optimale n_0.

Règle des 50 % /LeP76, BGLP75/.

Pour diverses valeurs des paramètres, et dans le cas où L est donnée par l'approximation de Belady et Kuehner, Leroudier et Potier /LeP76/ ont tabulé A_0 et A_1 $(= A_0/L(\frac{C}{n})\sigma_1)$ en fonction de n. Il s'avère que, lorsque A_0 atteint son maximum, le taux d'utilisation du DP, A_1, s'observe aux alentours de 50 %. Cela a conduit ces auteurs à préconiser la règle de contrôle suivante :

on active ou on désactive un programme selon que le taux d'utilisation du DP, mesuré pendant une période de temps récente, est inférieur ou non à 0.5.

Règle L=S. /DeK76/.

Cette heuristique consiste à régulariser le système de telle sorte que, en moyenne, la durée de vie des programmes soit soit un peu plus longue que la durée d'un transfert de page. Intuitivement, un taux de défauts de page plus élevé que le taux de service au DP risque d'embouteiller le système au niveau du DP, tandis qu'un taux de défauts trop petit peut indiquer une sous-utilisation du système.

Règle de l'injection aléatoire /GeK76/.

Gelenbe et Kurinckx /GeK76/ ont proposé le principe de contrôle suivant qui permet, outre une prévention des étranglements, un partage équitable des ressources entre programmes concurrents : chaque fois qu'un programme a utilisé un certain quantum de temps processeur, il est automatiquement suspendu et libère l'espace en mémoire centrale qu'il occupait ; il ne sera réinjecté dans le système qu'après un certain laps de temps (aléatoire dans /GeK76/) déterminé en fonction de la charge actuelle du système. L'intuition selon laquelle la durée de suspension doit être d'autant plus longue que le nombre global de programmes est élevé, a été confirmée, tant par une analyse approchée, que par des experiences de simulation.

Bien que ces méthodes de régulation de la charge du système semblent assez raisonnables et généralement confirmées par des études numériques, aucune d'elles n'a pu être justifiée d' d'une façon rigoureuse et précise. A notre connaissance, le seul cas particulier où une approximation analytique de n_0 a été donnée est celui d'un réseau cyclique sans unité d'ES (m=0). Avec l'approximation (5.7) de Belady et Kuehner, et pour autant que $1/f >> 1/\sigma_1$, Brandwajn et al [BGLP73] ont montré que n_0 peut être approximé par :

$$n_0 \simeq (a\sigma_1)^{1/k} C/e,$$

e désignant la base des logarithmes naturels. Avec cette approximation, il est intéressant de noter que la taille optimale d'une partition vaut : $C/n_0 = e(a\sigma_1)^{-1/k}$, et la durée de vie : $L(C/n_0) = e^k/\sigma_1$, indépendamment de la mémoire totale disponible C.

5.3. Partage optimal des ressources entre 2 programmes.

L'analyse précédente suppose des programmes identiques, et par conséquent néglige les aspects décisionnels relatifs au partage de la mémoire centrale et à l'ordonnancement des programmes à chacune des unités du système. D'une façon plus précise, étant donné un ensemble de programmes à exécuter, il s'agit de déterminer les règles de partage de la MC et les règles de priorité de service, de façon à maximiser le taux d'utilisation A_0 du processeur central.

En ce qui concerne l'allocation de la mémoire, il a été constaté qu'un partage biaisé, qui favorise arbitrairement certains programmes par rapport à d'autres, est généralement meilleur qu'un partage équitable où chacun des programmes reçoit la même quantité d'espace [Bel67, BeK69]. Une interprétation de ce phénomène, basée sur la concavité de la fonction durée de vie, mais qui ne tient pas compte des phénomènes d'attente à l'intérieur du système, a été donnée dans [DeS73]. Quant au problème de la coordination temporelle des tâches à l'intérieur d'un réseau fermé, il n'a pu être analysé que pour certaines structures particulièrement simples.
Il se dégage de ces résultats, notamment de celui de Spirn [Spi76], qu'on doit donner la priorité aux programmes les moins sujets à des défauts de page. Nous allons illustrer ces principes dans le cas de 2 programmes, de fonctions durée de vie connues, alternant entre un PC et un DP.

5.3.1. Le modèle.

Désignons par L_1 et L_2 les fonctions durée de vie des deux programmes, et par C le nombre total de cadres de page disponibles en mémoire centrale. Un partage de la mémoire est spécifié par un couple d'entiers (c_1, c_2) tel que :

$$c_1, c_2 \geq 0, \quad \text{et} \quad c_1 + c_2 = C ;$$

c_i (i = 1, 2) représente le nombre de cadres alloués au programme i.

Etant donné un partage (c_1, c_2), nous supposerons que, pour chacun des 2 programmes, les durées de temps virtuel entre 2 défauts de page sont distribuées exponentiellement, de taux :

$$f_1 = 1/L_1(c_1), \quad \text{et} \quad f_2 = 1/L_2(c_2)$$

respectivement. La durée d'un transfert de page au DP est supposée être une exponentielle de moyenne $1/\sigma$.

Sous un partage donné, l'état du système à un instant quelconque est spécifié par les programmes dans la file du PC à cet instant. Il y a évidemment 4 états, à savoir : "1", "2", "1+2" et "$\emptyset$". Chaque fois que le système passe par les états "1+2" ou "$\emptyset$", où les 2 programmes sont à une même unité de service, il faut choisir lequel à être servi. Une politique d'ordonnancement est donc décrite par les probabilités de choix : dp(i), pc(i) ; i =1,2, dp(i) représentant la probabilité pour que le programme i soit servi au DP alors que les 2 programmes sont au DP (état "$\emptyset$"), et pc(i) la probabilité pour que, le système étant dans l'état "1+2", le programme i soit servi au PC.

En supposant les L_i connues, nous allons déterminer les valeurs de c_i, dp(i), pc(i) de façon à maximiser le taux d'utilisation du processeur central.

5.3.2. Politiques d'ordonnancement optimales.

Au cours du temps, le PC est alternativement inactif et occupé. Quelle que soit la politique d'ordonnancement, une période d'inactivité dure le temps d'un transfert de page, donc $1/\sigma$ en moyenne. Le taux d'utilisation du PC vaut donc :

$$A_0 = B / (B + 1/\sigma), \qquad (5.10)$$

B représentant la durée moyenne d'une période d'occupation du PC. Maximiser A_0 revient donc à maximiser B.

Pour $S \neq \emptyset$, désignons par B(S) le délai moyen pour que le système passe de l'état S à l'état $\emptyset$. Une période d'occupation commence par l'état "i", i = 1, 2, si et seulement si le programme i a été choisi pour recevoir le service au DP. Par conséquent :

$$B = dp(1)B(1) + dp(2)B(2). \qquad (5.11)$$

Lorsque le système est dans l'état "1", le délai jusqu'au prochain changement d'état, minimum entre une exponentielle de taux f_1 et une exponentielle de taux σ, est une exponentielle de taux $f_1+\sigma$. Ce changement d'état peut correspondre :

- soit à un défaut de page du programme 1, auquel cas la période d'occupation du PC se termine (événement de probabilité $f_1(f_1+\sigma)^{-1}$)
- soit à une fin de pagination pour le programme 2, et le nouvel état sera "1+2".

Par conséquent :

$$B(1) = (f_1+\sigma)^{-1} + \sigma(f_1+\sigma)^{-1}B(1+2). \qquad (5.12)$$

D'une façon similaire :

$$B(2) = (f_2+\sigma)^{-1} + \sigma(f_2+\sigma)^{-1}B(1+2). \qquad (5.13)$$

Supposons, pour la commodité, que :

$$f_1 \leq f_2. \qquad (5.14)$$

Il résulte alors immédiatement des 2 dernières équations :

$$B(1) \geq B(2). \qquad (5.15)$$

Ainsi, quel que soit l'ordonnancement au PC, il faut accorder, au DT, la priorité préemptive au programme ayant le plus petit taux de défauts de page ($dp(1) = 1$).

Il reste maintenant à rechercher les probabilités $pc(i)$ qui maximisent $B = B(1)$. En vertu de (5.12), cela revient à maximiser $B(1+2)$.

Lorsque les 2 programmes sont simultanément au PC,

- si le programme 1 est servi, alors, après un temps moyen de $1/f_1$, on sera dans l'état "2",
- sinon, après un temps moyen de $1/f_2$, on sera dans l'état "1".

Par conséquent :

$$B(1+2) = pc(1)(1/f_1 + B(2)) + pc(2)(1/f_2 + B(1)).$$

Remplaçons dans cette équation $B(1)$ et $B(2)$ par leurs expressions données par (5.12) et (5.13). Nous obtenons, après quelques simplifications triviales :

$$B(1+2) = 1/f_1 + 1/f_2 + \sigma/f_1f_2, \qquad (5.16)$$

indépendamment de $pc(1)$, $pc(2)$.

Ainsi, le taux d'utilisation du PC ne dépend pas de la règle d'ordonnancement au PC.

La valeur optimale de ce taux est donnée par (5.10), où B vaut, lorsque $f_1 \leq f_2$:

$$\begin{aligned} B &= B(1) \\ &= (f_1+\sigma)^{-1} + \sigma(f_1+\sigma)^{-1}(1/f_1 + 1/f_2 + \sigma/f_1f_2) \\ &= 1/f_1 + \sigma/f_1f_2. \end{aligned} \tag{5.17}$$

5.3.3. Partage optimal de la mémoire.

Nous supposerons, pour la facilité, que les 2 programmes ont même fonction durée de vie : $L_1 = L_2 = L$. Naturellement, L est une fonction monotone non décroissante de c :

$$L(c_1) \geq L(c_2), \text{ pour } c_1 \geq c_2. \tag{5.18}$$

Nous supposerons également que L est une fonction concave, ce qui est généralement vrai pour de grandes valeurs de c :

$$L(c_1+1) - L(c_1) \leq L(c_2+1) - L(c_2), \text{ pour } c_1 \geq c_2. \tag{5.19}$$

Par raison de symétrie, nous pouvons nous restreindre aux partages tels que $c_1 \geq c_2$. En vertu de l'analyse faite au § 5.3.2, la recherche d'un partage optimal de la mémoire centrale consiste à trouver un entier c_1 qui maximise :

$$B(c_1; C) = L(c_1) + \sigma L(c_1)L(c_2), \tag{5.20}$$

sous les contraintes :

$$c_1 \geq c_2 = C-c_1 \geq 0. \tag{5.21}$$

Pour une valeur donnée de C, désignons par $\bar{c}_1(C)$ la valeur optimale de c_1. Nous montrerons que, lorsque la mémoire totale est de C+1 cadres, alors le nombre optimum de cadres à allouer

au programme 1 s'obtient immédiatement par l'équation de récurrence :

$$\bar{c}_1(C+1) = \begin{cases} \bar{c}_1(C) + 1, \text{ si } B(\bar{c}_1(C)+1;\ C+1) \geq B(\bar{c}_1(C);\ C+1), \\ \bar{c}_1(C), \text{ dans le cas contraire.} \end{cases} \tag{5.22}$$

Pour alléger la démonstration, nous poserons $\bar{c}_1 = \bar{c}_1(C)$.

1) Pour tout $c_1 \geq \bar{c}_1$ ($c_2 = C-c_1 \leq C-\bar{c}_1 = \bar{c}_2$) :

$$B(c_1+1;\ C+1) = L(c_1+1) + \sigma L(c_1+1)L(c_2)$$
$$= L(c_1) + \sigma L(c_1)L(c_2) + (L(c_1+1)-L(c_1))(1+\sigma L(c_2))$$

Or :

$L(c_1) + \sigma L(c_1)L(c_2) \leq L(\bar{c}_1) + \sigma L(\bar{c}_1)L(\bar{c}_2)$, par définition de $\bar{c}_1$;

$L(c_1+1)-L(c_1) \leq L(\bar{c}_1+1)-L(\bar{c}_1)$, en vertu de (5.19) ;

$L(c_2) \leq L(\bar{c}_2)$, en vertu de (5.18).

Par conséquent :

$$B(c_1+1;\ C+1) \leq L(\bar{c}_1+1) + \sigma L(\bar{c}_1+1)L(\bar{c}_2) = B(\bar{c}_1+1;\ C+1).$$

2) Pour tout $c_1 \leq \bar{c}_1$ ($c_2 \geq \bar{c}_2$) :

$$B(c_1;\ C+1) = L(c_1) + \sigma L(c_1)L(c_2+1)$$
$$= L(c_1) + \sigma L(c_1)L(c_2) + \sigma L(c_1)(L(c_2+1)-L(c_2)).$$

Comme au point 1) :

$L(c_1) + \sigma L(c_1)L(c_2) \leq L(\bar{c}_1) + \sigma L(\bar{c}_1)L(\bar{c}_2)$;

$L(c_2+1)-L(c_2) \leq L(\bar{c}_2+1)-L(\bar{c}_2)$;

$L(c_1) \leq L(\bar{c}_1)$.

Dès lors :

$$B(c_1;\ C+1) \leq B(\bar{c}_1;\ C+1).$$

En vertu des points 1) et 2), il résulte immédiatement que :

$$\max(B(\bar{c}_1+1;\ C+1),\ B(\bar{c}_1;\ C+1)) \geq B(c_1';\ C+1),$$

pour tout $c_1' = 0,\ldots,C+1$,

ce qui établit l'équation (5.22).

Cette propriété permet trivialement la détermination des partages optimaux de la mémoire centrale, et cela successivement pour C = 1, 2, 3, etc,... Les partages ainsi obtenus sont généralement biaisés, notamment lorsque la durée moyenne $1/\sigma$ d'une pagination est longue comparée à la durée de vie.
La condition $B(\bar{c}_1+1;\ C+1) \geq B(\bar{c}_1;\ C+1)$, sous laquelle $\bar{c}_1(C+1) = \bar{c}_1+1$, peut être réécrite sous la forme :

$$dL_1 \geq \sigma(L_1 dL_2 - L_2 dL_1), \qquad (5.23)$$

où : $L_i = L(\bar{c}_i)$,

$dL_i = L(\bar{c}_i+1) - L(\bar{c}_i)$, $i = 1, 2$.

Le processus de partage commencera donc à accorder des cadres au programme 1 jusqu'à ce que la condition (5.23) n'est plus satisfaite : l'amélioration dL_1 est alors trop petite pour justifier l'allocation d'un cadre supplémentaire au programme 1. L'allocation d'un cadre au programme 2 a pour effet de laisser dL_1 inchangé et de diminuer $\sigma(L_1 dL_2 - L_2 dL_1)$, ce qui tend à favoriser de nouveau la condition (5.23). Les cadres seront alors alloués au programme 1, et ainsi de suite.

L'analyse de ce paragraphe s'étend difficilement au cas de plus de 2 programmes. D'une façon générale, la recherche d'une politique d'ordonnancement stationnaire optimale peut être formulée comme un problème d'optimisation d'une fonction rationnelle de plusieurs variables. Dans le cas de 3 programmes, numérotés de telle sorte que $f_1 \leq f_2 \leq f_3$, nous donnons ici la politique optimale. La démonstration, beaucoup plus fastidieuse que dans le cas de 2 programmes, est laissée au lecteur.

Ordonnancement optimal de 3 programmes :

Au DP, la priorité préemptive doit être accordée aux programmes de taux de défauts de page les plus petits.

Au PC, l'ordre de service est quelconque, sauf dans les 2 cas où

- les programmes 1 et 2 sont au PC, et
- les programmes 1 et 3 sont au PC.

Dans ces 2 cas, il faut traiter le programme 1 d'abord.

Sous une telle politique, la valeur (maximale) de la durée moyenne d'une période d'occupation du PC vaut :

$$B = 1/f_1 + \sigma/f_1f_2 + \sigma^2/f_1f_2f_3 + \sigma^2(f_2-f_1)/(f_1+\sigma)f_1f_2f_3.$$

La recherche des règles de priorité optimales dans un réseau fermé sera discutée plus en détail au paragraphe suivant, où nous proposerons une approche basée sur la programmation dynamique.

5.4. Règles de priorité optimales au disque de pagination -- Une approche heuristique.

5.4.1. Motivation d'une approche par programmation dynamique.

Le problème de la recherche de politiques d'ordonnancement optimales dans un réseau fermé de serveurs exponentiels a été esquissé au § 5.3.2, et cela dans un cas des plus simples. Dans un contexte plus général, une formulation mathématique du problème peut être la suivante.

Si l'on suppose les durées de service des programmes à chacune des stations distribuées exponentiellement et indépendantes entre elles, et les probabilités d'acheminement (routing probabilities) des programmes décrites par une chaîne de Markov, alors l'état du réseau à tout instant peut être spécifié par les identités des programmes à chaque file d'attente du système. S'il y a en tout m stations et n programmes, le nombre total des états est évidemment m^n. Pour chaque état où plusieurs programmes sont en attente à une même unité de service, il faut sélectionner le programme à prendre en charge par l'unité en question. Ces choix peuvent être spécifiés par des probabilités de choix, dont l'ensemble constitue une politique d'ordonnancement particulière. Sous une polique donnée, les probabilités des états du système en régime asymptôtique sont liées par des relations de la forme (équations de balance) : $P = X.P$, où P représente le vecteur des probabilités d'état, et X est une matrice dont chaque élément dépend linéairement des probabilités de choix de la politique. Pour que P soit univoquement déterminé, il y a lieu d'ajouter

une relation exprimant que la somme des probabilités de tous les états vaut 1.

Prenons comme objectif la maximisation du taux d'utilisation d'une certaine unité "fondamentale" du réseau, le PC par exemple. Ce taux, qui s'obtient en sommant les probabilités sur tous les états correspondants à une file non vide à l'unité en question, est clairement une fonction rationnelle des probabilités de choix de la politique, variables de notre problème d'optimisation. Une telle méthode a été utilisée par Spirn /Spi76/ pour le problème d'ordonnancement du § 5.3. Pour des structures plus générales (plus de 2 programmes, par exemple), elle conduit à des équations vite insolubles analytiquement.

Nous proposons dans ce paragraphe une approche heuristique basée sur la théorie de la programmation dynamique. Traiter le problème comme un problème d'optimisation à horizon fixe et à temps discret permet de déterminer les meilleurs choix instantanés d'une façon récursive. Dans le contexte du réseau décrit au paragraphe suivant, où l'accent est mis surtout sur les phénomènes d'étranglement au niveau du DP, nous montrerons qu'il faut accorder la priorité préemptive aux programmes de durées de vie les plus longues. Nous montrerons également que cette politique minimise le taux d'utilisation du DP.

5.4.2. Formulation de l'ordonnancement optimal dans un réseau cyclique.

Considérons un système constitué, d'une part, d'un DP, et d'autre part, d'un certain nombre d'unités d'activité. Le DP

est considéré comme une station ne pouvant servir qu'un seul programme à la fois. Les autres unités du système, à savoir le PC et les unités d'ES, sont dénommées unités d'activité. Nous les supposerons en nombre suffisant pour que tout programme actif (c'est-à-dire qui n'est pas dans la file d'attente du DP) soit toujours pris en charge par l'une d'elles. En d'autres termes, le DP est considéré comme le seul goulot d'étranglement possible du système, et l'ensemble des autres unités comme une station comportant une infinité de serveurs en parallèle.

Dans ce réseau, circule un ensemble N de programmes, d'une façon cyclique. Tous les programmes ont même durée moyenne de service au DP, soit $1/\sigma$. Néanmoins, ils peuvent différer quant à leur durée d'activité. Désignons par $1/f_j$ la durée d'activité moyenne du programme j, $j \in N$. Toutes ces durées sont supposées indépendantes entre elles et distribuées exponentiellement.

Nous nous proposons de rechercher les règles de priorité au niveau du DP qui maximisent l'activité totale des programmes pendant une période de temps donnée. Désignons par $A(S; T)$ l'activité moyenne maximale pendant une période (t_0, t_0+T), étant donné que les programmes appartenant à S ($S \subset N$) sont actifs à l'instant t_0 (N-S représente alors l'ensemble des programmes au DP à cet instant). Evidemment :

$$A(S; 0) = 0, \text{ pour tout } S. \tag{5.24}$$

Pour $T > 0$, en vertu du principe d'optimalité, ces activités satisfont aux équations fonctionnelles :

$$A(S; T) = |S|.dt + o(dt) + X + Y + Z, \tag{5.25}$$

où : dt représente un pas temporel positif quelconque,

o(dt) une quantité telle que $o(dt)/dt \to 0$, lorsque $dt \to 0$,

et où :

$$X = \begin{cases} \sigma dt. \max_{j \notin S} A(S+j;\ T-dt), & \text{si } S \neq N, \\ 0, & \text{si } S = N ; \end{cases} \tag{5.26}$$

$$Y = \begin{cases} \sum_{i \in S} f_i dt.A(S-i;\ T-dt), & \text{si } S \neq \emptyset \\ 0, & \text{si } S = \emptyset ; \end{cases} \tag{5.27}$$

$$Z = \begin{cases} (1 - (\sigma + \sum_{i \in S} f_i)dt).A(S;\ T-dt), & \text{si } S \neq N \text{ et } S \neq \emptyset, \\ (1 - \sigma dt).\ A(\emptyset;\ T-dt), & \text{si } S = \emptyset, \\ (1 - \sum_{i \in N} f_i dt).A(N;\ T-dt), & \text{si } S = N. \end{cases} \tag{5.28}$$

Notons que le choix du programme j à servir au DP (éq. (5.26)) est effectué à chaque instant, de sorte que cette formulation tient compte des politiques non stationnaires dans le temps.

Nous substituons à ce problème un problème à pas discret de la façon suivante. Nous négligerons le terme o(dt) dans (5.25), et prendrons T multiple entier de dt, T = K.dt. Nous poserons alors :

$$A(S, K) = A(S;\ T). \tag{5.29}$$

5.4.3. Détermination des politiques maximisant l'activité.

Théorème 5.1.

> Pour tout horizon K, l'ordonnancement qui maximise l'activité consiste à accorder la priorité préemptive aux programmes de durées d'activité moyennes les plus longues. En termes plus précis, pour tout S, si j et k désignent 2 programmes dans N-S tels que :
>
> $$f_k \leq f_j, \tag{5.30}$$

alors :

$$A(S+k, K) \geq A(S+j, K), \text{ pour tout } K \geq 0. \qquad (5.31)$$

Démonstration.

L'inégalité à démontrer est triviale pour K = 0 (5.24). Supposons-la vraie jusqu'à K-1. En vertu des éq. (5.25-28) et de l'approximation (5.29), nous avons :

$$A(S+k, K) - A(S+j, K) = dX + dY + dZ,$$

où :

$$dX = \sigma dt.(\max_{i \notin S+k} A(S+k+i, K-1) - \max_{i \notin S+j} A(S+j+i, K-1)),$$

$$dY = \sum_{i \in S+k} f_i dt.A(S+k-i, K-1) - \sum_{i \in S+j} f_i dt.A(S+j-i, K-1),$$

$$dZ = (1 - (\sigma + \sum_{i \in S+k} f_i)dt).A(S+k, K-1) - (1 - (\sigma + \sum_{i \in S+j} f_i)dt).A(S+j, K-1).$$

Pour évaluer le signe de ces différences, nous distinguerons 2 cas :

1er cas : $\max_{i \notin S+j} A(S+j+i, K-1) = A(S+j+k, K-1)$.

Dans ce cas, puisque :

$$\max_{i \notin S+k} A(S+k+i, K-1) \geq A(S+k+j, K-1)$$

(le choix du programme particulier j n'est pas nécessairement optimal), nous avons clairement :

$$dX \geq 0.$$

Réécrivons dY sous la forme :

$$dY = \sum_{i \in S} f_i dt.(A(S+k-i, K-1) - A(S+j-i, K-1)) + (f_k - f_j)dt.A(S, K-1).$$

Par induction : $A(S+k-i, K-1) \geq A(S+j-i, K-1)$, pour tout $i \in S$.

Par conséquent :

$$dY \geq (f_k - f_j)dt.A(S, K-1). \qquad (5.32)$$

En ce qui concerne dZ, nous avons par induction :

$$A(S+k, K-1) \geq A(S+j, K-1),$$

de sorte que :

$$dZ \geq (f_j - f_k)dt.A(S+j, K-1). \qquad (5.33)$$

Collectant les inégalités (5.32) et (5.33), nous obtenons :

$$dY+dZ \geq (f_j - f_k)dt.(A(S+j, K-1) - A(S, K-1)).$$

La différence $f_j - f_k$ est positive ou nulle, par hypothèse.

Au lemme 5.1, nous démontrerons que $A(S+j, K-1) - A(S, K-1) \geq 0$.

Il en résulte donc finalement : $dY+dZ \geq 0$.

$2^{ème}$ cas : $\max_{i \notin S+j} A(S+j+i, K-1) = A(S+j+\bar{I}, K-1)$, $\bar{I} \neq k$.

Nous avons alors :

$$dX \geq \sigma dt.(A(S+k+\bar{I}, K-1) - A(S+j+\bar{I}, K-1))$$

(le choix de $\bar{I}$ dans N-S-k est sous-optimal)

$$\geq 0 \text{ (par induction)}.$$

La démonstration de l'inégalité $dY+dZ \geq 0$ se fait exactement comme dans le 1^{er} cas. C.Q.F.D.

Lemme 5.1.

Pour tout S, pour tout $j \notin S$, et pour tout $K \geq 0$:

$$A(S+j, K) - A(S, K-1) \geq 0. \qquad (5.34)$$

Démonstration.

La démonstration se fait également par induction. Nous avons:

$$A(S+j, K) - A(S, K-1) = 1.dt + dX' + dY' + dZ',$$

où :

$$dX' = \sigma dt.(\max_{i \notin S+j} A(S+j+i, K-1) - \max_{i \notin S} A(S+i, K-1)),$$

$$dY' = \sum_{i \in S} f_i dt.(A(S+j-i, K-1) - A(S-i, K-1)) + f_j dt.A(S, K-1),$$

$$dZ' = (1 - (\sigma + \sum_{i \in S+j} f_i)dt).A(S+j, K-1)$$
$$- (1 - (\sigma + \sum_{i \in S} f_i)dt).A(S, K-1).$$

Pour démontrer que dX' est positive ou nulle, nous distinguerons encore 2 cas.

- Si $\max_{i \notin S} A(S+i, K-1) = A(S+j, K-1)$, alors, pour tout $i \notin S+j$:

  $A(S+j+i, K-1) - A(S+j, K-1) \geq 0$, par induction.

Par conséquent $dX' \geq 0$.

- Si $\max_{i \notin S} A(S+i, K-1) = A(S+\bar{I}, K-1)$, avec $\bar{I} \neq j$, alors :

  $dX' \geq \sigma dt.(A(S+j+\bar{I}, K-1) - A(S+\bar{I}, K-1))$

  ≥ 0, par induction.

D'autre part, en ce qui concerne dY', l'hypothèse d'induction implique trivialement que :

$$dY' \geq f_j dt.A(S, K-1). \qquad (5.35)$$

Dans dZ', nous avons également : $A(S+j, K-1) \geq A(S, K-1)$, de sorte que :

$$dZ' \geq -f_j dt.A(S, K-1). \qquad (5.36)$$

Les inégalités (5.35) et (5.36) impliquent finalement :

$dY'+dZ' \geq 0$. C.Q.F.D.

5.4.4. Détermination des politiques minimisant l'utilisation du disque de pagination.

L'approche précédente peut évidemment être utilisée dans le cas d'un objectif autre que la maximisation de l'utilisation des unités d'activité du système. Dans le cas de notre modèle où le seul goulot d'étranglement se situe au niveau du DP, un objectif secondaire pourrait être la minimisation de l'utilisation

de ce DP. Pour cet objectif, nous allons montrer que la règle "du programme de plus longue durée d'activité d'abord" est aussi optimale. Ce résultat confirme l'intuition selon laquelle, sous une charge donnée, les unités d'activité du système (PC, ES) sont utilisées au mieux lorsque, en contre partie, l'unité de pagination est sollicitée le moins souvent possible.

Comme précédemment, nous subdivisons l'horizon d'optimisation en pas dt suffisamment petits pour pouvoir négliger les termes en dt d'ordre supérieur à 1. Désignons par D(S, K) l'utilisation moyenne minimale du DP sur un horizon de K pas, S représentant l'ensemble des programmes actifs à l'instant "initial". En vertu du principe d'optimalité, ces D(S, K) satisfont aux équations :

$$D(S, 0) = 0, \text{ pour tout } S, \tag{5.37}$$

et, pour $K > 0$:

$$D(S, K) = \Theta dt + U + V + W, \tag{5.38}$$

où :

$$\Theta = \begin{cases} 1, & \text{si } N-S \neq \emptyset \quad \text{(il existe des programmes au DP)} \\ 0, & \text{si } N-S = \emptyset; \end{cases} \tag{5.39}$$

$$U = \begin{cases} \alpha dt. \min\limits_{j \notin S} D(S+j, K-1), & \text{si } N-S \neq \emptyset, \\ 0, & \text{si } N-S = \emptyset; \end{cases} \tag{5.40}$$

$$V = \begin{cases} \sum\limits_{i \in S} f_i dt.D(S-i, K-1), & \text{si } S \neq \emptyset \\ 0, & \text{si } S = \emptyset; \end{cases} \tag{5.41}$$

$$W = \begin{cases} (1 - (\sigma + \sum_{i \in S} f_i)dt).D(S, K-1), & \text{si } S \neq N \text{ et } S \neq \emptyset \\ (1 - \sigma dt).D(\emptyset, K-1), & \text{si } S = \emptyset \\ (1 - \sum_{i \in N} f_i dt).D(N, K-1), & \text{si } S = N. \end{cases} \qquad (5.42)$$

La démonstration de la propriété anoncée nécessite d'abord le lemme suivant, analogue du lemme 5.1 :

Lemme 5.2.

> Pour tout S, pour tout $j \notin S$, et pour tout $K \geq 0$:
> $D(S, K) - D(S+j, K) \geq 0$.

Ce lemme permet alors de démontrer le

Théorème 5.2.

> Pour tout S, pour tout couple de programmes j et k dans N-S tel que $f_k \leq f_j$, nous avons :
> $D(S+j, K) - D(S+k, K) \geq 0$, pour tout $K \geq 0$.

Les raisonnements utilisés au § 5.4.3 pour la démonstration du lemme 5.1 et du théorème 5.1 se transposent ici sans aucune difficulté. Cette transposition sera laissée aux soins du lecteur.

L'approche par programmation dynamique, illustrée dans ce paragraphe, peut s'appliquer évidemment à d'autres structures de réseaux, pour autant que l'objectif à optimiser soit une fonction additive du temps. Dans notre exemple particulier, l'utilisation totale d'une certaine unité pendant un horizon fixé est additive en ce sens qu'elle est la somme des utilisations sur tous les intervalles de temps élémentaires qui constituent l'horizon. Des recherches personnelles sont en cours dans le cas de systèmes où les programmes diffèrent entre eux du point de vue de leurs caractéristiques d'acheminement (routing) à l'intérieur du réseau.

5.6. Conclusions.

Le comportement d'un système informatique peut être modélisé par un réseau de files d'attente dans lequel circulent des programmes, processus demandeurs de ressources. Depuis les travaux de Jackson /Jac63/ et de Gordon et Newell /GoN67/, l'étude analytique exacte de tels réseaux a connu de nomdreux développements, notamment avec Buzen /Buz71/, Reiser et Kobayashi /ReK74/, Basket et al /BCMP75/. Ainsi, les probabilités à l'état stationnaire du système ont été obtenues dans les cas où chaque station de service du réseau appartient à l'un des 4 types suivants :

1) La station est constituée d'un seul serveur. La durée de service, la même pour chaque programme, est distribuée exponentiellement. Le taux de service peut dépendre de la longueur de la file à cette station. La discipline de service est FIFO.

2) Un seul serveur partage son temps uniformément entre tous les programmes en file (discipline "processor sharing"). Les durées de service sont quelconques, et peuvent différer d'un programme à l'autre.

3) La station est constituée d'une infinité de serveurs, et les durées de service des programmes sont quelconques, comme en 2).

4) Les programmes sont traités par un seul serveur suivant une discipline LIFO préemptive. Aucune restriction n'est imposée aux durées de service des différents programmes.

Pour d'autres structures de réseaux, diverses méthodes approchées ont été proposées, notamment la méthode de décomposition /Cou71, Bra74/, et l'approximation par les processus de diffusion /Gav68, Gel73b, Kob74/.

A côté de ces résultats, assez généraux certes, mais uniquement descriptifs du comportement du système, se posent de nombreux problèmes décisionnels relatifs au partage optimal des ressources.

Un premier problème est d'étudier comment varie l'efficacité du système, mesurée par le taux d'utilisation de son PC, en fonction du nombre de programmes concurrents. De nombreuses applications numériques des résultats généraux confirment l'intuition selon laquelle le système s'étrangle au delà d'une charge optimale n_0. Au stade actuel des recherches, il semble que seule une approche numérique permette de dégager certains critères approximatifs selon lesquels l'optimalité est atteinte.

Un deuxième problème concerne les règles de priorité des programmes, de caractéristiques opérationnelles différentes, aux diverses unités du système. Généralement, la recherche d'une politique d'ordonnancement optimale peut être formulée comme un problème de maximisation d'une fonction rationnelle de plusieurs variables. Pour des modèles les plus simples, il s'avère qu'on doit accorder la priorité aux programmes les moins sujets à des défauts de page.

Un dernier problème qui a été esquissé est de déterminer comment la mémoire centrale doit être partitionnée entre un ensemble de programmes. Ce niveau d'optimisation est tributaire du niveau précédent puisque le taux d'utilisation du PC à maximiser dépend des règles de priorité adoptées. Un partage biaisé peut être meilleur qu'une équipartition. Au stade des

recherches actuelles, c'est un problème ouvert.

Les deux principaux niveaux de contrôle soulevés dans ce chapitre, à savoir le réglage de la charge du système et l'ordonnancement des programmes dans le réseau, sont essentiellement axés sur la maximisation de l'utilisation des unités "effectives" du système : le processeur et les unités d'entrée-sortie. Pour des systèmes interactifs, une voie intéressante de recherche pourrait être orientée vers des critères plus "humains", par exemple des temps de réponse brefs ou une équité de service parmi les différents programmes.

REFERENCES

ADU71 Aho, A. V., Denning, P. J., and Ullman, J. D., "Principles of optimal page replacement," J. ACM 18, 1 (01-1971), 80-93.

BGLP73 Brandwajn, A., Gelenbe, E., Lenfant, J., and Potier, D., "A model of program and system behavior in virtual memory," Tech. Rept., IRIA-Laboria, France (10-1973).

BGLP75 Badel, M., Gelenbe, E., Leroudier, J., and Potier, D., "Adaptative optimization of a time-sharing system's performance," Proc. IEEE : Special Issue on Interactive Computer Systems (06-1975), 958-965.

BCMP74 Baskett, F., Chandy, K. M., Muntz, R. R., Palacios, F.G., "Open, closed, and mixed networks of queues with different classes of customers," J. ACM 22, 2 (04-1975), 248-260.

BaM76 Batson, A. P., and Madison, A. W., " Measurements of major locality phases in symbolic reference strings," Proc. Int. Symp. on Computer Performance Modeling, Measurement, and Evaluation (Chen & Franklin, Eds), Harvard U. (03-1976), 75-84.

Bel66 Belady, L. A., "A study of replacement algorithms for a virtual storage computer," IBM Syst. J. 5, 2 (1966), 78-101.

Bel67 Belady, L. A., "Biased replacement algorithms for multiprogramming," IBM T. J. Watson Research Center Note NC697 (03-1967).

BeK69 Belady, L. A., and Kuehner, C. J., "Dynamic space sharing in computer systems," C. ACM 12, 5 (05-1969), 282-288.

BNS69 Belady, L. A., Nelson, R. A., and Shedler, G. S., "An anomaly in space-time characteristics of certain programs running in a paging environment," C. ACM 12, 6 (06-1969), 349-353.

Bra74 Brandwajn, A., "A model of a time-sharing virtual memory system solved using equivalence and decomposition methods," Acta Informatica 4 (1974), 11-47.

BrG68 Brawn, B. S., and Gustavson, F. G., "Program behavior in a paging environment," Proc. AFIPS 1968 FJCC, 33, II (1968), 1019-1032.

Buz71 Buzen, J. P., "Queueing network models of multiprogramming," Ph. D. Thesis, Harvard University (1971).

Buz73 Buzen, J. P., "Computational algorithms for closed queueing networks with exponential servers," C.ACM 16,9 (1973), 527-531.

Buz76 Buzen, J. P., "Fundamental laws of computer system performance," Proc. Int. Symp. on Computer Performance, Harvard U. (03-1976), 200-210.

CFL73 Chamberlin, D. D., Fuller, S. H., and Liu, L. Y., "An analysis of page allocation strategies for multiprogramming systems with virtual memory," IBM J. Res. Dev. 17 (1973), 404-412.

ChL74 Chang, A, and Lavenberg, S. S., "Work rates in closed queueing networks with general independent servers," Oper. Res. 22 (1974), 838-847.

ChO72 Chu, W. W., and Opderbeck, H., "The page fault frequency replacement algorithm," Proc. AFIPS 1972 FJCC, 41, I (1972), 597-609.

CoD73 Coffman, E. G. Jr, and Denning, P. J., "Operating Systems Theory," Prentice-Hall (1973), chapt. 6-7.

CoV76 Courtois, P. J., and Vantilborgh, H., "A decomposable model of program paging behaviour," Acta Informatica 6 (1976), 251-275.

Com67 Comeau, L. W., "A study of the effect of user program optimization in a paging system," ACM Symp. on Operating System Principles, Gatlinburg, Tenn. (1967).

Cou71 Courtois, P. J., "On the near complete decomposability of networks of queues and of stochastic models of multiprogramming systems," Carnegie-Mellon Univ., Computer Science Dept., Research Report (11-1971).

Den68a Denning, P. J., "The working set model for program behavior," C. ACM 11, 5 (05-1968), 323-333.

Den68b Denning, P. J., "Resource allocation in multiprocess computer system, " MIT Project MAC , Rept. No. MAC-TR-50, Cambridge, Mass. (1968).

Den70 Denning, P. J., "Virtual Memory," Computing Surveys 2, 3 (09-1970), 153-189.

DeS72 Denning, P. J., and Schwartz, S. C., "Properties of the working set model," C. ACM 15, 3 (03-1972), 191-198.

DeS73 Denning, P. J., and Spirn, J. R., "Dynamic storage partitioning," ACM Op. Syst. Rev. 7, 4 (10-1973), 73-77.

DeG75 Denning, P. J., and Graham, G. S., "Multiprogrammed memory management," Proc. IEEE: Special Issue on Interactive Computer Systems (06-1975), 924-939.

DeK76 Denning, P. J., and Kahn, K. C., "An L=S criterion for optimal multiprogramming," Proc. Int. Symp. on Computer Performance Modeling, Measurement, and Evaluation, Harvard U. (03-1976), 219-229.

DeS76 Denning, P. J., and Slutz, D. R., "Generalized working set and optimal measures for segment reference strings," Techn. Rpt., Purdue U. (02-1976).

DeT76 Denning, P. J., and Trân-Quôc-Tê, "On the optimality of working set policies," Techn. Rpt., Purdue U. (03-76).

Gav68 Gaver, D. P., "Diffusion approximations for certain congestion problems," J. Appl. Prob. 5 (1968), 607-623.

Gel73a Gelenbe, E., "An unified approach to the evaluation of a class of replacement algorithms," IEEE Trans. on Comp. C-22, 6 (1973), 611-618.

Gel73b Gelenbe, E., "Modèles de Systèmes Informatiques," Thèse de Doctorat d'Etat, Univ. Paris VI (1973).

GTB73 Gelenbe, E., Tiberio, P., and Bookhorst, J. C. A., "Page size in demand-paging systems," Acta Informatica 3 (1973), 1-23.

GLP74 Gelenbe, E., Lenfant, J., et Potier, D., "Analyse d'un algorithme de gestion simultanée mémoire centrale-disque de pagination," Acta Informatica 4 (1974), 321-346.

GeK76 Gelenbe, E., and Kurinckx, A., "Random injection control of multiprogramming in virtual memory," Modeling and Performance Evaluation of Comp. Syst. (Gelenbe, ed.), North-holland Publishing Company (1976).

GoN67 Gordon, W. J., and Newell, G. F., "Closed queueing systems with exponential servers," Oper. Res. 15 (1967), 254-265.

HaG71 Hatfield, D. J., and Gerald, J., "Program restructuring for virtual memory," IBM Syst. J. 10, 3 (1971), 168-192.

Hat72 Hatfield, D. J., "Experiments on page size, program access patterns, and virtual memory performance," IBM J. of Res. and Dev. 16, 1 (01-1972), 58-66.

InK74 Ingargiola, G., and Korsh, J. F., "Finding optimal demand paging algorithms," J. ACM 21, 1 (01-1974),40-53.

Jac63 Jackson, J. R., "Jobshop-like queueing systems," Management Science 10 (1963), 131-142.

Ker69 Kernighan, B. W., "Optimal segmentation points for programs," Proc. 2nd Symp. on Operating Systems Principles, Princeton U. (10-1969), 47-53.

KELS62 Kilburn, T., Edwards, D. G. B., Lanigan, M. J., and Summer, F. H., "One-level storage system," IRE Trans. E. C. 11, 2 (04-1962), 223-235.

Kob74 Kobayashi, H, "Application of the diffusion approximation to queueing networks : Part 1 - Equilibrium queue distributions," J. ACM 21 (1974), 316-328.

KuL69 Kuck, D. J., and Lawrie, D. H., "The use and performance of memory hierarchies," Computer and Information Sciences II (Tou, Ed.), Academic Press (1969).

Len74a Lenfant, J., "Evaluation sur des modèles de comportement de programme de la taille d'un ensemble de travail," RAIRO (bleu) B2 (1974), 77-92.

Len74b Lenfant, J., "The delay network model for program behaviour," Computer Architecture and Networks (Gelenbe & Mahl, Eds), 299-330.

Len74c Lenfant, J., "Comportement des programmes dans leur espace d'adresse. Application à la gestion des mémoires hiérarchisées," Thèse de Doctorat, Univ. de Rennes (1974).

LeB75 Lenfant, J., et Burgevin, P., "Empirical data on program behavior," Proc. Int. Computing Symposium (E. Gelenbe & D. Potier, Eds), 163-169.

LeP76 Leroudier, J., and Potier, D., "Principles of optimality for multiprogramming," Proc. Int. Symp. on Computer Performance Modeling, Measurement, and Evaluation, Harvard U. (03-1976), 211-218.

MaB76 Madison, A. W., and Batson, A. P., "Characteristics of program localities," C. ACM 19, 5 (05-1976), 285-294.

MGST70 Mattson, R. L., Gecsei, J., Slutz, D.R., Traiger, I.L., "Evaluation techniques for storage hierarchies," IBM Syst. J. 9, 2 (1970), 78-117.

Mit74 Mitra, D., "Some aspects of hierarchical memory systems," J. ACM 21, 1 (01-1974), 54-65.

Mor73 Morrison, J. E., "User program performance in virtual storage systems," IBM Syst. J. 12, 3 (1973), 216-237.

OpC74 Opderbeck, H., and Chu, W. W., "Performance of the page fault frequency replacement algorithm in a multiprogramming environment," Information Processing 74, North-Holland P. C. (1974), 235-241.

PrF76 Prieve, B. G., and Fabry, R. S., "VMIN - an optimal variable space page replacement algorithm," C. ACM 19, 5 (05-1976), 295-297.

RaK68 Randell, B., and Kuehner, C. J., "Demand paging in perspective," Proc. AFIPS 1968 FJCC, 33, II (1968), 1011-1018.

ReK75 Reiser, M., and Kobayashi, H., "Queueing networks with multiple closed chains," IBM J. of Res. and Dev. (05-1975), 283-294.

Say69 Sayre, D., "Is automatic folding of programs efficient enough to displace manual ?," C. ACM 12, 12 (12-1969), 656-660.

ShT72 Shedler, G. S., and Tung, C., "Locality in page reference strings," SIAM J. Computing 1, 3 (09-1972), 218-241.

Spa74 Spaniol, O., "Demand prepaging algorithms basing on a model of locality of programs," Computer Architecture and Networks (Gelenbe & Mahl, Eds), 515-527.

SpD72 Spirn, J. R., and Denning, P. J., "Experiments with program locality," Proc. AFIPS 1972 FJCC, 41 (1972), 611-621.

Spi76 Spirn, J. R., "Multi-queue scheduling of two tasks," Proc. Int. Symp. on Comp. Perf., Harvard U. (03-1976), 101-108.

ThI72 Thorington, J. M. Jr, and Irwin, J. D., "An adaptative replacement algorithm for paged-memory computer systems," IEEE Trans. on Comp. C-21, 10 (1972), 1053-1061.

TQT75a Trân-Quôc-Tê, "Some principles of page forecasting," Rapport de rech., Fac. Univ. N.-D. de la Paix, Namur (04-1975).

TQT75b Trân-Quôc-Tê, "Performances of least reference probability paging algorithms under locality in paging behavior," Proc. GI 75 Conf., Dortmund (1975), 715-735.

TQT75c Trân-Quôc-Tê, "An open formulation of working set policies," Rapport de Rech., Fac. Univ. N.-D. de la Paix, Namur (12-1975).

TQT76a Trân-Quôc-Tê, "A heuristic model for analysis of memory use under static partition allocation strategies," Proc. Int. Symp. on Computer Performance Modeling, Measurement, and Evaluation, Harvard Univ. (03-1976), 272-281.

TQT76b Trân-Quôc-Tê, "Space time duality in virtual storage allocation," Rapport de Rech., Fac. Univ. N.-D. de la Paix, Namur (04-1976).

TQT76c Trân-Quôc-Tê, "L'optimisation à terme immédiat dans la pagination des programmes localisés," RAIRO (bleu) suppl. au vol. B10 : Modèles, Mesures, et Simulation des Systèmes Informatiques (05-1976), 59-82.

Yu73 Yu, P. L., "Introduction to domination structures in multicriteria decision problems," Multiple Decision Making (Cochrane & Zeleny, Eds) Univ. of South Carolina Press (1973), 249-261.