A propos de la simulation d'un système informatique

Claude Boksenbaum

THESE

présentée à

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

POUR OBTENIR LE GRADE DE

DOCTEUR D'ETAT

Claude BOKSENBAUM

A PROPOS DE LA SIMULATION D'UN SYSTEME INFORMATIQUE

Thèse soutenue le 16 septembre 1975 devant la Commission d'Examen

Président	N. GASTINEL
Rapporteur	L. BOLLIET
	S. KRAKOWIAK
	R. MAHL
Examinateur	F. SALLE

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

M. Michel SOUTIF	Présidents	M. Louis NEEL
M. Gabriel CAU	Vice-Présidents	MM. Lucien BONNETAIN
		Jean BENOIT

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Paul	Chimie
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Clinique de Pédiatrie et Puériculture
	BERNARD Alain	Mathématiques Pures
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BEZES Henri	Pathologie chirurgicale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BOUCHERLE André	Chimie et toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARLIER Georges	Biologie végétale
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Clinique Oto-Rhino-Laryngologique
	CHATEAU Robert	Thérapeutique (Neurologie)
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBERMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELORMAS Pierre	Pneumo-Phtisiologie
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée

MM.	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DRUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de dermatologie et syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques pures
	GALVANI Octave	Mathématiques pures
	GASTINEL Noël	Mathématiques appliquées
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	KAHANE André	Physique Générale
	KLEIN Joseph	Mathématiques pures
	KOSZUL Jean-Louis	Mathématiques pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Mlle	LUTZ Elisabeth	Mathématiques pures
	MALGRANGE Bernard	Mathématiques pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	MULLER Jean-Michel	Thérapeutique (néphrologie)
	NEEL Louis	Physique du solide
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY-PEYROULA Jean-Claude	Physique
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	RINALDI Renaud	Physique
	DE ROUGEMONT Jacques	Neuro-chirurgie
	SEIGNEURIN Raymond	Microbiologie et hygiène
	SENGEL Philippe	Zoologie
	SIBILLE Robert	Construction mécanique
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
MM.	VERAIN André	Physique
	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale

PROFESSEURS ASSOCIES

MM.	CHEEKE John	Thermodynamique
	COPPENS Philip	Physique
	CORCOS Gilles	Mécanique
	CRABBE Pierre	CERMO
	GILLESPIE John	I.S.N.
	ROCKAFELLAR Ralph	Mathématiques appliquées

PROFESSEURS SANS CHAIRE

Mlle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBROISE-THOMAS Pierre	Parasitologie
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BERTRANDIAS Jean-Paul	Mathématiques pures
	BIAREZ Jean-Pierre	Mécanique
	BILLET Jean	Géographie
Mme	BONNIER Jane	Chimie générale
MM.	BOUCHET Yves	Anatomie
	BRUGEL Lucien	Energétique
	CONTE René	Physique
	DEPASSEL Roger	Mécanique des fluides
	GAUTHIER Yves	Sciences biologiques
	GAUTRON René	Chimie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	GROULADE Joseph	Biochimie médicale
	HACQUES Gérard	Calcul numérique
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Méd. Préventive
	IDELMAN Simon	Physiologie animale
	JOLY Jean-René	Mathématiques pures
	JULLIEN Pierre	Mathématiques appliquées
Mme	KAHANE Josette	Physique
MM.	KUHN Gérard	Physique
	LOISEAUX Jean	Physique nucléaire
	LUU-DUC-Cuong	Chimie organique
	MAYNARD Roger	Physique du solide
	PELMONT Jean	Biochimie
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
Mlle	PIERY Yvette	Physiologie animale
MM.	RAYNAUD Hervé	Mathématiques appliquées
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	RICHARD Lucien	Biologie végétale
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
MM.	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	AMBLARD Pierre	Dermatologie
	ARMAND Gilbert	Géographie
	ARMAND Yves	Chimie
	BARGE Michel	Neurochirurgie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamique
M.	BOUCHARLAT Jacques	Psychiatrie adultes
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BRODEAU François	Mathématiques (IUT B)
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARDON Michel	Géographie
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CORDONNIER Daniel	Néphrologie
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DELOBEL Claude	M.I.A.G.
	DENIS Bernard	Cardiologie
	DOUCE Roland	Physiologie végétale
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FONTAINE Jean-Marc	Mathématiques pures
	GAUTIER Robert	Chirurgie générale
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques appliquées
	GROS Yves	Physique (stag.)
	GUITTON Jacques	Chimie
	HICTER Pierre	Chimie
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	KOLODIE Lucien	Hématologie
	KRAKOWIAK Sacha	Mathématiques appliquées
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LEROY Philippe	Mathématiques
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)
	MICHOULIER Jean	Physique (IUT A)
Mme	MINIER Colette	Physique
MM.	NEGRE Robert	Mécanique
	NEMOZ Alain	Thermodynamique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PERRET Jean	Neurologie
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAMBAUD Pierre	Pédiatrie
Mme	RENAUDET Jacqueline	Bactériologie
MM.	ROBERT Jean-Bernard	Chimie-Physique

MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean-Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VROUSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	COLE Antony	Sciences nucléaires
	FORELL César	Mécanique
	MOORSANI Kishin	Physique

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

MM.	BOST Michel	Pédiatrie
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	FAURE Gilbert	Urologie
	MALLION Jean-Michel	Médecine du travail
	ROCHAT Jacques	Hygiène et hydrologie

Fait à Saint Martin d'Hères, OCTOBRE 1974.

"MEMBRES DU CORPS ENSEIGNANT DE L'I.N.P.G."

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Radioélectricité
	BESSON Jean	ELectrochimie
	BONNETAIN Lucien	Chimie Minérale
	BONNIER Etienne	Electrochimie, Electrométallurgie
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	ELectronique
	COUMES André	Radioélectricité
	FELICI Noël	Electrostatique
	PAUTHENET René	Physique du solide
	PERRET René	Servomécanismes
	SANTON Lucien	Mécanique
	SILBER Robert	Mécanique des fluides

PROFESSEUR ASSOCIE

M.	BOUDOURIS Georges	Radioélectricité

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du solide et cristallographie
	COHEN Joseph	Electrotechnique
	DURAND Françis	Metallurgie
	MOREAU René	Mécanique
	POLOUJADOFF Michel	ELectrotechnique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM.	BOUVARD Maurice	Génie mécanique
	CHARTIER Germain	Electronique
	FOULARD Claude	Automatique
	GUYOT Pierre	Chimie minérale
	JOUBERT Jean Claude	Physique du solide
	LACOUME Jean Louis	Géophisique
	LANCIA Roland	Physique atomique
	LESPINARD Georges	Mécanique
	MORET Roger	Electrotechnique nucléaire
	ROBERT François	Analyse numérique
	SABONNADIERE Jean Claude	Informatique fondamentale et appliquée
Mme	SAUCIER Gabrièle	Informatique fondamentale et appliquée

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan Doré	Automatique

CHARGE DE FONCTIONS DE MAITRES DE CONFERENCES

M.	ANCEAU François	Mathématiques appliquées

Fait à Saint Martin d'Hères JANVIER 1974

à la mémoire de
Jean Luc Haour

Je remercie Monsieur GASTINEL, d'une part d'avoir bien voulu présider le jury, et d'autre part pour ses remarques concernant la rédaction de cette thèse.

Je remercie Monsieur BOLLIET pour la confiance qu'il m'a témoignée, ainsi que pour son soutien tout au long de mon travail.

Je remercie également Messieurs MAHL et KRAKOWIAK, pour l'attention critique qu'ils ont manifesté et Monsieur SALLE pour avoir bien voulu faire partie du jury.

Je remercie sincèrement tous les chercheurs avec qui j'ai pu travailler et qui ont largement contribué à l'évolution de mes idées, sans que cela fût clair au moment même.

Je remercie enfin ceux qui ont contribué à la réalisation matérielle de cet ouvrage.

TABLE DES MATIERES .1.

CHAPITRE I

SCIENCES, MODELES ET MESURES

> ... Il est beaucoup plus intéressant de regarder là où l'on ne va pas pour la bonne raison que là où on va, il sera toujours temps d'y regarder quand on y sera.
>
> J.ROUXEL
> Les Shadoks et les Gibis

Les informaticiens se demandent si l'informatique est ou sera une science. Améliorer, mesurer les produits informatiques matériels ou programmés c'est leur attribuer une valeur bonne ou mauvaise.

Ces activités posent donc les problèmes philosophiques de la "Connaissance". Comment situer les activités informatiques dans ce cadre plus général ? Il me paraît indispensable d'essayer de le faire. Bien qu'une part essentielle du problème de la Connaissance consiste à s'interroger sur sa possibilité, sa nature et ses limites, nous supposerons que le concept de connaissance objective est assez clair pour nous intéresser plus spécialement aux relations entre les mathématiques, les sciences expérimentales et les techniques.

1.1. CLASSIFICATIONS DES SCIENCES ET ROLE DES MODELES

Les mathématiques, qui n'ont acquis leur fondement qu'au cours de ce siècle (cf. Bourbaki [17]), est la science des "objets" ou "êtres" abstraits. Il est indéniable néanmoins que le développement de ses branches a été influencé par l'homme, les sciences expérimentales ou les techniques. Par exemple, le développement de l'algèbre de Boole, de l'analyse numérique, des grammaires formelles, des algorithmes de tri, a été influencé par l'emploi des ordinateurs. En ce sens, l'informatique contribue au développement des mathématiques.

Les autres sciences, que nous appellerons "expérimentales", étudient les objets concrets et leur comportement que nous appellerons "phénomènes physiques". Elles combinent la description de ces phénomènes (l'expérience) et une tentative d'explication cohérente. Cette explication consiste à identifier les objets concrets et leurs relations à des êtres et relations abstraites (que nous appellerons leurs "modèles") qui soient cohérents mathématiquement. Ainsi, chaque explication justifie simultanément l'adéquation des mathématiques (axiomes et schémas) au monde réel et les modèles proposés dans l'étude du phénomène correspondant. Par exemple, le paradoxe de Zénon et d'Elée : "A un instant donné, la flèche n'est ni devant elle, ni derrière elle ; elle est à sa place, donc immobile", reflète des mathématiques mal adaptées au monde réel puisque l'instant y est défini mais qu'une "succession" d'instants ne peut produire le mouvement. Même de nos jours on peut se demander comment la notion de vitesse est assimilée : quelle est la vitesse moyenne d'un cycliste qui va de A à B à 30 km/heure et revient de B à A à 20 km/heure ?

Le manque d'unité des sciences expérimentales, qui nous montre l'immensité du champ restant à défricher, est dû à l'aspect indépendant des modèles proposés pour l'étude des divers phénomènes. De la matière inerte où les interactions rencontrées sont "simples", à la matière vivante où les interactions sont asservies [64], puis au cerveau humain où les niveaux d'asservissement s'imbriquent de façon partiellement inconnue [53], jusqu'aux groupements humains (groupes et sociétés), les chaînons manquants se mettent en place.

Je pense que ces modèles, potentiellement compatibles, continueront leur unification vers une science expérimentale unique, comme l'ont fait ceux de la Physique et de la Chimie, ou ceux de diverses branches de la physique (théorie des quanta et électromagnétisme, par exemple).

Cette unification est un souci constant de chaque science expérimentale qui, d'une part, introduit de nouveaux modèles convenant à des phénomènes jusqu'alors inexpliqués et veut, d'autre part, conserver la cohérence de toutes les explications. Celle-ci s'obtient, le plus souvent, en démontrant que pour les phénomènes qu'ils expliquent, les modèles précédemment utilisés constituent de bonnes approximations des nouveaux modèles. Par exemple, la théorie classique de la propagation de la lumière en ligne droite, qui explique bien la formation d'images dans certains systèmes optiques (lentilles), est une approximation de la théorie ondulatoire en ce qui concerne ces phénomènes. De même, la théorie de la gravitation newtonienne se déduit, en tant qu'approximation de la théorie de la relativité d'Einstein.

L'enseignement ne met pas en lumière la convergence des divers modèles. En mathématique, on utilise d'une façon anticipée des résultats qui ne seront plus justifiés même lorsque le moment le permettra (qu'est-ce-que la preuve par 9 ?)- Plus grave encore, le lien entre certaines notions peut être escamoté : peu de gens trouveront difficile de calculer l'aire d'un rectangle par une intégrale, alors que l'assimilation d'une aire à une intégrale se fait à l'aide d'un rectangle ! (Dans cet exemple, il faudrait montrer que certaines propriétés de l'aire (additivité, inclusion) conduisent aux propriétés caractéristiques des intégrales).

L'évolution des modèles des sciences expérimentales pose la question fondamentale de leurs domaines de validité. Or, un modèle est justifié profondément par la présomption que "des effets identiques peuvent avoir des causes identiques". Cette présomption, renforcée par la répétition d'expériences dans les mêmes conditions ou dans des conditions voisines, ne permet pas de fixer le domaine de validité du modèle.
Au contraire, l'affirmation que "des effets différents (ou assez différents) ont des causes différentes", équivalent logique de la relation "la cause entraîne l'effet", sert à réfuter la validité d'un modèle pour l'explication d'un phénomène. Ainsi ce domaine de validité d'un modèle est délimité par des points qui lui sont extérieurs, là où d'autres modèles sont nécessaires.

Les techniques, méthodes et procédés, utilisent des phénomènes physiques pour créer et, à première vue, ne s'embarrassent pas forcément de savoir si une explication cohérente de ces phénomènes existe.

Ainsi, on a perfectionné la bicyclette pendant plus de cinquante ans sans connaître le facteur essentiel de sa stabilité directionnelle qu n'a été établi qu'en 1970 [44]. De même, certains produits pharmaceutiques sont fabriqués sans qu'aucune théorie n'explique leurs propriétés.

1.2. OBSERVATION DES PHENOMENES - ROLE DES MESURES

L'observation des phénomènes est une partie fondamentale des sciences expérimentales et des techniques. Cette observation peut révéler que des modèles déjà définis s'appliquent bien (et constituent ainsi une vérification ou "validation"), ou que les modèles existants expliquent mal le phénomène.

Cette seconde conclusion de remise en cause nécessite toujours un examen minutieux des conditions de l'observation qui peut en être la première responsable. Par exemple, l'hypothèse de la "génération spontanée" apparamment confirmée par l'apparition de germes dans un bouillon de culture rendu stérile, s'appuyait sur une observation où le milieu atmosphérique voisin n'était pas stérilisé, comme l'a démontré Pasteur. De même "l'horreur du vide" qu'aurait eu la nature, s'est trouvée désarmée devant l'expérience de Pascal au Puy de Dôme. Néanmoins, un doute excessif à l'égard de telles observations est profondément ancré dans notre civilisation : la théorie de la rotation de la terre autour du soleil qui explique parfaitement le mouvement des astres se heurta à des traditions établies comme les récits bibliques ou mythologiques et, après que Copernic puis Galilée l'aient démontrée il lui faudra plus de cent ans pour être reconnue !.

De nos jours, les "objets volants non identifiés" ou "la transmission de pensée" qui ont été souvent observés sans être encore expliqués, après avoir été méprisés par la science établie, font maintenant l'objet d'un recensement qui sera certainement utile lorsque des modèles essaieront de les expliquer.

Je crois que la généralisation de cette attitude d'enregistrement systématique des observations faites sera un des facteurs essentiels de la progression future des sciences expérimentales.

Lorsque les conditions d'observations de phénomènes non explicables ont été vérifiées et se montrent convenables il est nécessaire soit d'introduire un modèle plus fin où les définitions élémentaires auront été reculées (comme par l'introduction de nouvelles particules en physique nucléaire ou de motivations non conscientes en psychologie par Freud), soit de réévaluer l'importance des divers facteurs qui entrent en jeu dans ces phénomènes. Cette dernière nécessité a pu être érigée en principe de négligeabilité par G. Bachelard [5] : "A tous les niveaux de son expérimentation

le physicien doit se poser la question constante : qu'est-ce-que je peux négliger ?"

Les observations devraient, idéalement, être : universelles (c'est-à-dire faisables par n'importe quel observateur), exhaustives (c'est-à-dire regroupant toutes les conditions de l'observation), précises (faisant apparaître les nuances).
Pour atteindre ce dernier objectif, on utilise la notion de mesure qui a pour but de quantifier l'observation. Cette quantification qui porte aussi bien sur la grandeur mesurée que sur la précision de la méthode, est basée sur la mathématique et plus particulièrement sur la statistique.

Il faut noter que la méthode de mesure peut s'appuyer sur d'autres sciences expérimentales, afin d'atteindre une précision suffisante. Par exemple, pour déterminer l'âge de certaines roches, certains fossiles ou ossements humains, on mesure la transformation isotopique des éléments. De plus, ces méthodes qui fournissent des "prothèses" à l'observation directe par l'homme essaient d'assurer l'universalité par l'intermédiaire d'instruments de mesure, dont Bachelard a pu dire qu'ils étaient de la "théorie matérialisée".

Les mesures s'identifient donc de plus en plus à l'observation, et contribuent à la vérification des modèles ou à leur mise en cause. Dans ce dernier cas ils constituent une base de données importantes pour les chercheurs qui doivent imaginer de nouveaux modèles.

1.3. SITUATION DE L'INFORMATIQUE - CRITERES DE PERFORMANCE

L'informatique apparaît comme la technique d'utilisation des ordinateurs. Les services rendus (par exemple maintenance et exploitation d'une banque de données, exécution d'un algorithme écrit en langage de haut niveau) sont très éloignés des contraintes physiques élémentaires. En fait l'informatique s'apparente à une technique d'organisation de l'activité intellectuelle à partir de techniques précédemment maîtrisées.

Cette technique induit de nouveaux problèmes de mathématiques et de nouvelles branches des mathématiques (algèbre de Boole, logique, approximation, langages). Elle induit aussi de nouveaux problèmes concernant l'activité intellectuelle de l'homme.

Néanmoins son but primordial est la réalisation de services pour les sociétés humaines. Ce but nécessite une adaptation des outils informatiques aux hommes et vice-versa [41]. Il nécessite aussi une prospection de nouveaux services afin de cerner les difficultés à vaincre et les conditions économiques dans lesquelles ces services seront rendus.

La résistance au changement, ou inertie de cette technique intellectuelle est moins importante que dans les autres techniques et favorise son évolution suivant des critères "scientifiques". A court terme, certaines solutions techniques semblent profitables à la fois aux utilisateurs et aux constructeurs, mais introduisent en fait un accroissement énorme de cette inertie. Par exemple, les machines "compatibles" avec les précédentes, ou celles d'autres constructeurs, le maintien de langages à la syntaxe peu claire et à l'utilisation pleine de pièges, constituent à mon avis, un danger à long terme. Les conversions ainsi retardées pourraient s'avérer plus coûteuses par la suite !

L'activité informatique tourne ainsi autour de trois pôles :

i) - activité mathématique : émergence et résolution de problèmes
ii) - amélioration de la technique d'organisation de services existants
iii) - recherche de nouveaux services à rendre et ébauche d'un prototype le faisant.

L'activité mathématique garde éternellement sa valeur et n'a pas besoin d'autre justification. Mais la concordance entre les recherches mathématiques et leur utilisation immédiate peut être de la plus grande importance pour la société.

L'amélioration de la technique est encore plus sujette au compromis économique suivant : l'apport d'une expérience surpasse-t-il son coût ?

De même l'apport d'une ébauche d'un prototype en tant que méthode nouvelle compense-t-il son coût , compte tenu soit des méthodes alternatives anciennes qui auraient pu être utilisées, soit de la conjoncture technologique qui conditionne le passage à un stade de grande diffusion?

On voit donc que les activités informatiques , autre que la pure mathématique, sont conditionnées par des critères économiques. Ces activités se résument très souvent à l'énumération de choix pris pour concevoir ou réaliser un produit, avec leurs justifications, a posteriori. Il serait profitable de mettre en lumière les divers compromis que l'on aura à fixer, avant la réalisation, afin de ne pas raffiner arbitrairement certaines parties. Notons que certains de ces compromis ne portent pas exclusivement sur des coûts immédiatement quantifiables comme des utilisations de ressources (mémoire contre temps d'unité centrale) mais font appel à des qualités subjectives :

- connaissance d'une technique plutôt que d'une autre
- facilité de mise au point, d'utilisation, de documentation
- fusion de concepts en un seul (esthétique).

La pleine conscience des compromis économiques auxquels elle doit faire face sera une preuve de la maturité de l'informatique. Cette attitude pourra s'appuyer sur les techniques d'évaluation des systèmes permettant leur amélioration.

Celle-ci comprend plusieurs étapes, d'une façon idéale :

1. détection d'un défaut de performances dans une réalisation utilisée, par des mesures
2. définition mathématique d'un remède et prédiction du gain de performances sous des conditions similaires à la réalité
3. réalisation des changements précédemment définis et analysés et contrôle de la performance atteinte par des mesures.

L'itération par retour aux étapes antérieures peut se produire :

- de la seconde vers la première si le manque de performances apparaît comme théoriquement irrémédiable (les exemples les plus simples se trouvant exprimés au chapître 2)

- de la troisième vers la seconde si des différences apparaissent, dues à des phénomènes négligés mathématiquement soit dans le modèle du fonctionnement soit dans le modèle des demandes de service (les chapîtres 4 et 5 donnant des exemples concrets de telles inquiétudes).

Le critère de performance, ou fonction à optimiser, combinera en général deux composantes : le coût matériel du service et la satisfaction que procure le service. Cette satisfaction peut comprendre l'attente (moyenne et variance), la sécurité, la disponibilité, l'aide à la conversion ou la compatibilité, l'aide à la mise au point, etc ..

Il serait illusoire de définir des combinaisons standardisées de ces composantes. Le nombre de situations particulières mettrait en défaut ces tentatives.

Par exemple, si l'on ne voit dans un système informatique qu'un outil sans possibilité de dialogue (cas de calculateurs spécialisés), l'aspect disponibilité ou sécurité primera sur les autres. Au contraire, si on utilise le système pour des travaux périodiques, la disponibilité cèdera la place au temps d'attente moyen. De même, si on l'utilise pour des travaux de recherche, la sécurité pourra s'effacer devant les outils d'aide à la mise au point.

Les systèmes à vocation générale veulent satisfaire un assez grand nombre de critères tout en étant incapables, au stade de la conception, de quantifier ceux-ci : cela peut aussi bien prouver soit que la discipline demeure très vivante, soit qu'elle n'en est qu'à ses balbutiements.

Il nous paraît plus essentiel de mettre en lumière les critères qui intéressent l'usager, leur sens de variation et les relations qu'ils peuvent avoir.

Avant d'aborder le problème de la simulation des systèmes informatiques, puis l'exemple de celle de CP 67 avec les résultats d'expériences, nous allons montrer comment des modèles dépouillés peuvent éclairer les causes profondes de l'inefficacité des systèmes, plus précisément l'inactivité des ressources physiques.

CHAPITRE 2

LES DEUX CAUSES PRINCIPALES D'INACTIVITE DES RESSOURCES DANS LES SYSTEMES INFORMATIQUES

L'activité des ressources physiques est la base d'un critère de performance assez naturel : celui d'un centre de calcul qui fait payer les services rendus d'après l'activité et le coût des ressources mises en jeu.

Or l'inactivité des ressources est engendrée par deux causes principales et indépendantes :

i) Tout d'abord, les travaux soumis concernent plusieurs ressources différentes et la charge totale a peu de chances de coïncider avec la capacité du système. Un modèle linéaire découlant d'une simplification d'un modèle de Mahl [60], montrera ce phénomène (cf. 8.1.). Dans un tel modèle, où les consommations de ressources par un travail sont supposées exemptes de contraintes de synchronisation, seules quelques ressources sont saturées (c'est-à-dire sont toujours actives).

ii) Ensuite, même les ressources que l'on pourrait s'attendre à être saturées subissent une inactivité lorsque les demandes de ressources deviennent des variables aléatoires. Sur un modèle simple nous montrerons ce phénomème (cf. 2.2.), ainsi que la manière dont le nombre d'usagers actifs (degré de multiprogrammation) tempère cette inactivité.

Le but de ce chapitre est de montrer que l'essence des phénomènes d'inactivité est assez simple pour que des modèles simples en rendent compte. Dans des modèles plus complexes on pourra essayer de distinguer la part de responsabilité de chacune des causes, soit analytiquement, soit par simulation.

2.1. Un modèle simple de système

Le système comprend n ressources indépendantes qui doivent assurer l'exécution de m tâches. La tâche i est caractérisée par sa demande, en durée de la ressource j, a_{ij}. Le système essaiera de déterminer combien de fois par unité de temps, f_i, il peut exécuter la tâche i. On note t_j le taux d'occupation de la ressource j. L'activité des ressources conduit aux relations suivantes :

$$\sum_{i=1}^{m} f_i a_{ij} = t_j \qquad j = 1,\ldots n \qquad (1)$$

ou, sous forme matricielle en notant A la matrice m x n d'élément a_{ij}, f le vecteur ligne (f_i), t le vecteur ligne (t_j)

$$f.A = t \qquad (1')$$

avec les contraintes :

$$f_i \geqslant 0,\ a_{ij} \geqslant 0,\ 0 \leqslant t_j \leqslant 1 \qquad (2)$$
$$i = 1,\ldots m, \quad j = 1,\ldots n$$

Dans ces relations, les inconnues peuvent être soit les fréquences, f_i, pour certaines tâches, soit les consommations de ressources, a_{ij}, de certaines autres. Le système d'équations est linéaire lorsque l'une de ces possibilités exclut l'autre pour chaque tâche.

Un critère de performance acceptable peut être :

$$F = \sum_{j=1}^{n} C_j t_j$$

où les C_j représentent le coût unitaire de la ressource j et F le coût d'occupation du système. Ce critère convient au chef d'une installation dont le prix de revient (achat ou location) est indépendant des travaux réalisés et où la ressource j sera facturée C_j par unité de temps.

Maximiser ce critère est un problème de "programmation linéaire"

$$\max\left(F = \sum_{j=1}^{n} C_j t_j\right) \qquad (3)$$

où les t_j satisfont aux relations (1) et aux contraintes (2), les variables associées à la tâche i étant soit f_i, soit $\{ a_{i1},\ldots,a_{in} \}$

Remarques :

1. Ce modèle néglige

. l'utilisation des ressources à des fins de gestion par le système d'exploitation ("overhead"),

. les contraintes de synchronisation dans l'utilisation des ressources par les tâches.

En conséquence, l'optimum est une borne supérieure des performances que le système réel pourra atteindre.

2. Certains changements n'altèrent pas la nature du problème :

. relations linéaires entre les tâches mises en jeu,

. impositions de limites sur les fréquences inconnues :

$$0 \leq \underline{f_i} \leq f_i \leq \overline{f_i}$$

. regroupement de plusieurs ressources en une seule ; par exemple si les ressources de 1 à j_1 sont en fait confondues en la ressource 1', en posant

$$t_{1'} = \sum_{j=1}^{j_1} t_j \qquad a'_{i1'} = \sum_{j=1}^{j_1} a_{ij} \tag{4}$$

on peut remplacer les relations 1 à j_1 par :

$$\sum_{j=1}^{j_1} \left(\sum_{i=1}^{m} f_i \, a_{ij} \right) = \sum_{i=1}^{m} f_i \, a_{i1'} = t_{1'} \tag{5}$$

avec les contraintes $0 \leq t'_1 \leq 1$ qui remplacent $0 \leq t_i \leq 1$ pour $i = 1$ à j_1, lorsque les ressources considérées sont "logiques" ou "fonctionnelles" celles-ci doivent être regroupées en ressources réelles pour le problème d'optimisation.

. L'influence d'autres ressources qui ne se partagent pas seulement que dans le temps, comme la mémoire, n'apparaît pas explicitement. Si cette influence peut être approchée par des relations linéaires, le problème ne change pas de nature.

Par exemple, la prise en compte de la mémoire centrale peut se faire, de façon simplifiée, à l'aide de l'intégrale dans le temps de son usage : si r_i est le temps de résidence en mémoire centrale de la tâche i qui occupe un volume de mémoire M_i, la taille totale de la mémoire étant M, on a la relation :

$$\sum_{i=1}^{m} M_i r_i f_i \leq M \tag{6}$$

Le temps de résidence peut être pris :

a) $r_i = \max_j (a_{ij})$

dans le cas où les demandes de ressource sont indépendantes temporellement que chacune nécessite la tâche en mémoire;

b) $r_i = \sum_{j=1}^{n} a_{ij}$

dans le cas où les demandes sont séquentielles.

La relation (6) suppose que seule la mémoire utilisée "activement" compte. En particulier, la durée de chargement des programmes (ou de la pagination) est négligée et tout programme en mémoire est actif.

Dans ces conditions, le cas a) n'est réaliste que si un programme utilise toute la mémoire ($M_i = M$, $i = 1, \ldots m$) ou si des programmes co-habitant en mémoire centrale utilisent des ressources disjointes (en particulier des unités centrales). Le cas b) correspond à une multiprogrammation généralisée à toutes les ressources.

Les fréquences trouvées peuvent être interprétées en nombre d'utilisateurs : lorsque les exécutions successives d'une tâche i correspondent à l'activité d'un utilisateur et que cette émission est effectuée à une fréquence fixe Φ_i, alors on peut interpréter $\frac{f_i}{\Phi_i}$, lorsque ce rapport est supérieur à 1, comme un nombre d'utilisateurs semblables que le système pourrait accueillir.

De nouvelles contraintes ou relations linéaires sur ce nombre d'utilisateurs peuvent être incorporées dans le modèle.

L'usage pratique du modèle est illustré par l'exemple suivant :

Soit un système constitué de trois ressources : UC, D1, D2, sur lequel trois tâches peuvent s'exécuter (la tâche 3 ayant une fréquence fixe de 1). Le tableau de consommation des ressources est le suivant :

	UC	D1	D2	Fréquence
tâche 1	0.05	0.125	0	f_1
tâche 2	0.1	0.1	0.25	f_2
tâche 3	0.5	0	0	1

correspondant aux équations :

$$0.05f_1 + 0.1f_2 + 0.5 = t_1$$

$$0.125f_1 + 0.1f_2 = t_2$$

$$0.25f_2 = t_3$$

avec les contraintes :

$$t_1, t_2, t_3, f_1, f_2 \geqslant 0$$

$$t_1, t_2, t_3 \leqslant 1$$

Ces équations et contraintes définissent un domaine convexe dont la projection sur le plan (f_1,f_2) s'obtient en éliminant t_1, t_2, t_3 des inéquations précédentes :

$$0 \leqslant 0.1f_1 + 0.2f_2 \leqslant 1$$

$$0 \leqslant 0.125f_1 + 0.1f_2 \leqslant 1$$

$$0 \leqslant 0.25f_2 \leqslant 1$$

$$f_1, f_2 \geqslant 0$$

ce qui donne le polygone hachuré de la figure suivante :

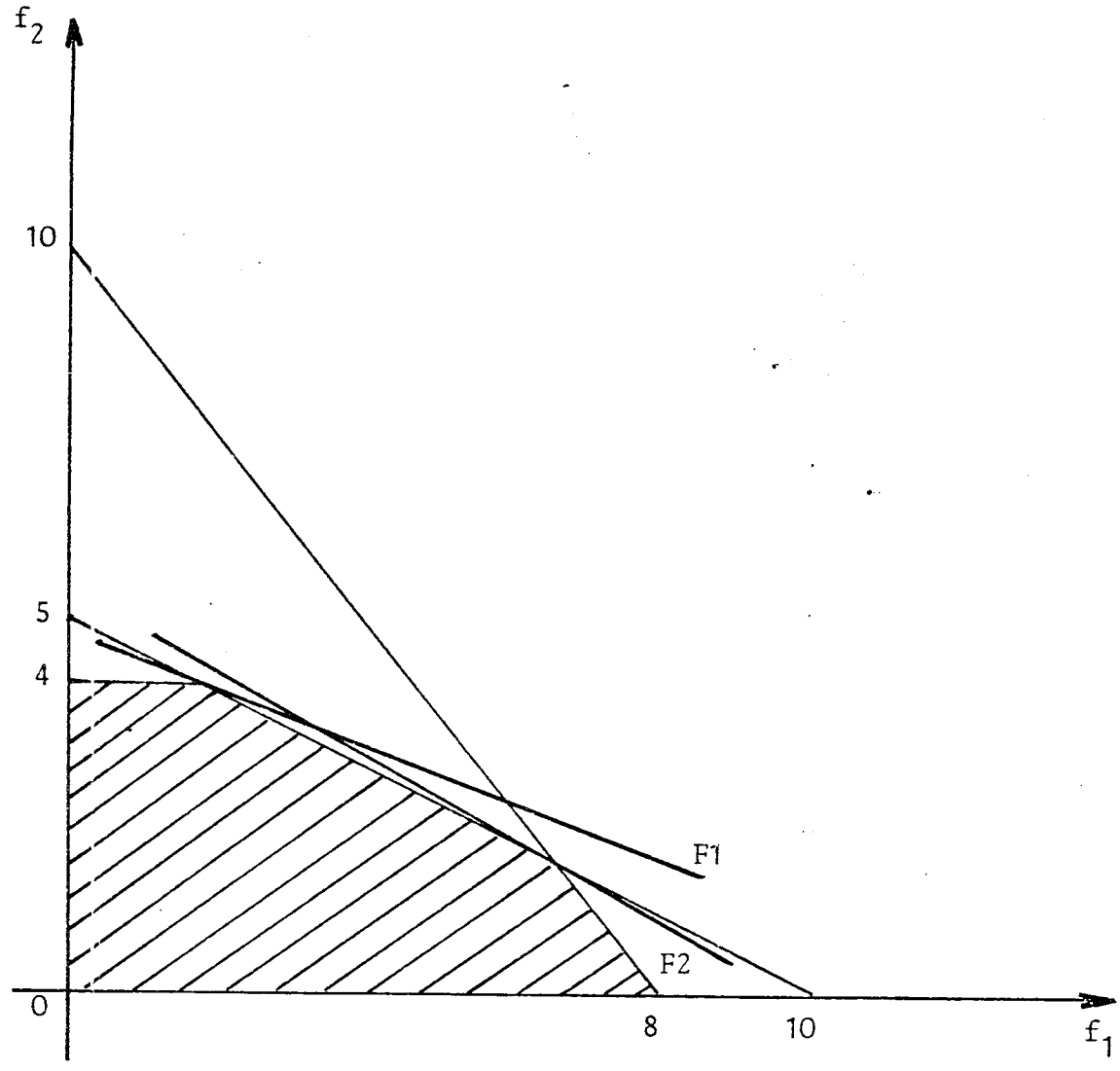


Figure 2.1.

Une valeur constante K du critère à maximiser

$F1 = 2t_1 + t_2 + t_3$

représente une famille d'hyperplans, de direction normale fixe. Leurs traces sur le plan (f_1,f_2) est une famille de droites

$0.225f_1 + 0.55f_2 = K - 1$

Le maximum correspond au(x) point(s) de contact(s) de la droite de cette famille tangente supérieurement au domaine convexe.

Dans notre cas, l'optimum correspond au point (2,4) du plan (f_1,f_2) qui donne :

$t_1 = 1$, $t_2 = 0.65$, $t_3 = 1$ et $F1 = 3.65$.

L'interprétation de ce résultat est qu'à l'optimum la ressource non saturée sera D1.

Pour le critère $F2 = 2t_1 + 2t_2 + t_3 = K$
correspondant à la droite $0.35t_1 + 0.65t_2 = K - 1$
l'optimum est au point $(\frac{20}{3},\frac{5}{3})$ du plan (f_1,f_2) qui donne

$t_1 = 1$, $t_2 = 1$, $t_3 = \frac{5}{12}$ et $F1 = 4 + \frac{5}{12} \neq 4.42$

D2 n'étant pas saturée.

On voit donc qu'un simple modèle linéaire permet de trouver une borne à l'activité de certaines ressources lorsque l'optimum est atteint. Lorsque cette borne est assez faible, on peut s'attendre à ce que dans la réalité plus complexe, où les hypothèses du modèle ne sont pas nécessairement vérifiées, la ressource envisagée subisse quand même une inactivité due au sous-phénomène étudié.

2.2. Effet de la variabilité de la demande

Considérons le système suivant, où m tâches demandent alternativement la ressource 1 puis la ressource 2, et les obtiennent dans l'ordr des demandes (FIFO) :

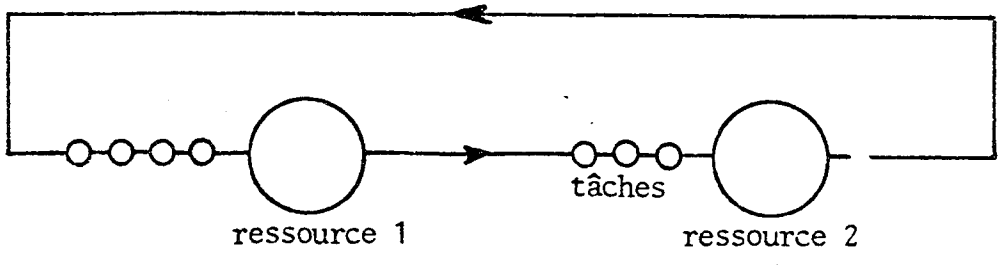


Figure 2.2.

Chaque tâche utilise la ressource 1 pendant un temps fixe a_1 et la ressource 2 pendant un temps fixe a_2. Si $a_1 > a_2$ la ressource 1 sera saturée (c'est-à-dire toujours en activité), tandis que la ressource 2 ne sera active que dans la proportion $\frac{a2}{a1}$.

Notons qu'au démarrage, et dans le cas où il n'y a qu'une seule tâche, ce résultat n'est pas valable. Rousset [73] a étudié entièrement le cas où les demandes dépendent des tâches, c'est-à-dire que la tâche i utilise $a_{1,i}$ de la ressource 1 et $a_{2,i}$ de la seconde.

Si nous supposons maintenant que les durées d'utilisation des ressources sont des variables aléatoires suivant des lois exponentielles, de moyenne a_1 et a_2 respectivement, nous avons un cas particulier de réseaux de serveurs exponentiels, étudiés par Jackson [43] Gordon & Newell [37].

Notons $r = \frac{a2}{a1}$ que nous supposerons < 1.

Alors l'activité de la ressource 1 en état stationnaire* sera (cf. appendice 1) :

$$A_1 = \frac{K_{m-1}}{K_m} \quad \text{où} \quad K_m = \sum_{k=0}^{m-1} r^k$$

c'est-à-dire ($r \neq 1$)

$$A_1 = \frac{1 - r^m}{1 - r^{m+1}} < 1$$

L'introduction d'un temps de service aléatoire a engendré une attente de la ressource "la plus saturée".

Le fait que A_1 croisse avec m (et l'on a même $\lim_{m \to \infty} A_1 = 1$) indique que le nombre de tâches dans le système compense cette attente due à un défaut de synchronisme. Cette compensation dépend de la saturation globale du système.

---

* C'est-à-dire la limite de la proportion de temps pendant laquelle la ressource sert un client.

Exemple :

Si l'on veut atteindre un taux d'activité de $1 - \varepsilon$ nous devrons avoir :

$$1 - \varepsilon = \frac{1 - r^m}{1 - r^{m+1}} \quad \text{ou} \quad r^m - (1 - \varepsilon)r^{m+1} = \varepsilon$$

$$(1 - (1 - \varepsilon)r)r^m = \varepsilon$$

soit : $m = \dfrac{\log \varepsilon - \log(1 - (1 - \varepsilon)r)}{\log r}$

soit pour $\varepsilon = 0.01$:

r	0.5	0.9	1
m	6	43	99

(pour $r = 1$, $A_1 = \frac{m}{m+1}$)

Ainsi, pour obtenir la saturation de la ressource 1 à 99 % il faudra respectivement 6, 43 ou 99 tâches suivant que r vaut 0.5, 0.9 ou 1, ce qui montre comment le nombre de tâches croit lorsque r tend vers 1.

D'une façon générale, dans un réseau fermé de serveurs exponentiels, le taux d'activité de ces serveurs augmente avec le nombre de tâches présentes dans le système, lorsque la durée de service de ces serveurs et les probabilités de transition sont constantes. Ce résultat, trivial en apparence, est démontré en appendice 1. Il devrait être vrai pour des réseaux à serveurs non exponentiels où l'état stationnaire existe [24].

Le phénomène précédent de "mauvaise synchronisation" se retrouve aussi dans des modèles déterministes. Ainsi, dans le modèle étudié par Rousset [73], où la tâche i demande respectivement $a_{1,i}$ et $a_{2,i}$ de ressource 1 et 2, s'il existe une tâche i_0 telle que

$$a_{1,i_0} + a_{2,i_0} > \max\left(\sum_{i=1}^{m} a_{1,i}, \sum_{i=1}^{m} a_{2,i}\right)$$

celle-ci induit une attente de ressources correspondant à la différence des deux membres de l'inégalité. Cette attente est due au fait que pendant que la tâche i_0 est en train d'être servie par une ressource, sa demande en autre ressource ne peut être prise en compte. En pratique, on pourra accroître le nombre de tâches dans le système, jusqu'à ce que la condition précédente ne soit plus valable.

La croissance du nombre de tâches qui diminue l'inutilisation des ressources milite en faveur de la mise en commun de celles-ci (cf. Kleinrock [49]). En fait, d'autres composantes du critère économique traduisant la satisfaction de l'usager (temps de réponse, disponibilité, accessibilité) limitent cette croissance.

CHAPITRE 3.

EXPRESSION DES MODELES DE SIMULATION

3.0. Introduction

Nous allons dégager les problèmes généraux d'expression des modèles de simulation que des langages spécialisés ont essayé de résoudre.

Tout d'abord, l'ordonnancement des actions dans le temps ne nous a pas semblé assez approfondi. Nous essaierons de dégager les conditions nécessaires et suffisantes de simulation temporelle des processus, après avoir rappelé la liaison entre algorithmes et processus et situé les relations que les systèmes informatiques ont avec ces concepts.

Ensuite nous verrons comment les problèmes d'expression algorithmique, de prise de mesures, d'initialisation, de spécification du modèle des "entrées" nous amènent à des caractéristiques de langages, soit le langage d'expression du modèle, soit ceux de contrôle conversationnel.

3.1. SIMULATEUR DE PROCESSUS PARALLELES

3.1.1. Processus et algorithmes

Le terme de "processus" est couramment utilisé dans les systèmes informatiques et le concept de processus a fait l'objet de tentatives de formalisation (cf. Horning & Randell [42]). Or il me semble que cette notion est proche de celle d'algorithme élaborée par Markov [62], et qu'il est intéressant de relier ces deux notions.

Un algorithme est la définition d'une application ou fonction par compositions d'applications "élémentaires". Cette définition est "constructive", car si l'on sait réaliser les fonctions élémentaires on saura réaliser l'application décrite par l'algorithme.

Plusieurs remarques vont relier cette notion d'algorithme aux mathématiques "classiques" et à l'informatique.

Remarque 1 : Unification des espaces de départ et d'arrivée des applications

Chacune des applications intervenant dans un algorithme est définie sur un espace commun à toutes qui est l'extension des espaces de départ et d'arrivée réellement utiles.

Par exemple, sur une machine ayant n registres et N mots qui peuvent prendre des valeurs dans I = [-V,V-1] intervalle des entiers relatifs, l'addition modulo 2Vest une application de I x I → I. Or l'adressage qui intervient dans toutes les instructions de ce type, en fait une application de $I^n \times I^N \to I^n \times I^N$ de la manière suivante :

$$I^n \times I^N \;\to\; I \times I \;\to\; I \;\to\; I^n \times I^N$$

sélection des opérandes — addition modulo V — immersion du résultat

en supposant qu'un des opérandes et le résultat se trouvent accueillis dans le même registre.

Ceci permet de faire dépendre les applications qui interviennent dans un algorithme du résultat des applications précédentes sans avoir de problèmes pour la définition des espaces de départ et d'arrivée. Cette dépendance est souvent exprimée par des "structures" conditionnelles, c'est-à-dire des applications du type h : si condition alors f sinon g.
Notons que l'on ne profitera de cette "séparation des cas" que si l'on effectue séquentiellement dans le temps les compositions successives.
Notons aussi que la structure conditionnelle peut être considérée comme un algorithme sur les fonctions élémentaires f,g (qui peuvent être des algorithmes) arbitraires dans un ensemble défini.

Les notations classiques d'applications ont donc été allégées dans l'étude des algorithmes, comme elles le sont aussi dans la théorie des langages formels, celle des graphes et celle des ensembles "flous". On peut se demander quel rôle ces simplifications jouent dans l'avancement ou la compréhension d'une théorie.

Notons aussi que l'espace commun sur lequel opèrent les algorithmes qui est la fermeture transitive d'un alphabet fini par concaténation, est dénombrable alors qu'en pratique il demeure fini.

Remarque 2 : Présentation des algorithmes

En général on n'énumère pas la séquence des applications qui composent un algorithme, mais on décrit d'une part un sous-ensemble d'applications (chacune constituant une "étape") et d'autre part la règle qui permettra de choisir quelle est la prochaine étape qui interviendra dans l'algorithme. C'est en particulier la présentation adoptée par Markov [62] ou celle des machines de Türing. Elle correspond aussi à l'usage courant (cf. Knuth [52] vol.1).

Cette règle, que l'on peut appeler "partie contrôle" nous intéressera lors de la synchronisation.

Remarque 3 : Influence des applications élémentaires sur la définition des algorithmes

La définition de l'algorithme est relative à un ensemble d'applications élémentaires ; de plus, chaque algorithme peut être utilisé comme une application élémentaire pour un nouvel algorithme. Ce fait est utilisé à la fois en théorie, par exemple pour définir un algorithme de multiplication entière à partir d'un algorithme de copie, et en pratique de façon intensive : à partir des instructions machine, des sous-programmes ou procédures sont définis qui serviront à la définition d'autres sous-programmes ou procédures.

A l'inverse, lorsqu'on fabrique ces sous-programmes, on ne sait pas, et l'on n'a pas besoin de savoir, si les fonctions élémentaires le sont réellement : celles-ci sont utilisées comme des "boîtes noires". Par exemple, on peut réaliser certaines instructions à l'aide de micro-programmes, comme on réalisait sur certains petits ordinateurs les instructions sur nombres à virgule flottante par programme.

Le principe de construction étagée ou fractionnée d'algorithme, à la base de la programmation "en couches" ou modulaire, ne fait que réaffirmer ce principe, sans rajouter quoi que ce soit de plus significatif, à mon avis.

De même, les langages de haut niveau définissent des structures de composition (appel de procédure, itérations, instructions conditionnelles) et des applications élémentaires plus agréables aux utilisateurs que les instructions machine. La compilation consiste à transformer un algorithme écrit dans ce langage en un autre utilisant les instructions machine qui donnera les mêmes résultats lorsque l'on ne s'intéresse qu'à l'espace des variables du premier.

Cette considération explique pourquoi lors d'une anomalie il est si embarrassant d'être confronté à l'état d'avancement du second algorithme ("dump" hexadécimal) alors qu'il serait judicieux de pouvoir, en un tel moment, revenir à la spécification initiale de l'algorithme.

On peut alors remarquer que les "résultats" attendus d'un algorithme sont en fait définis par les moyens d'accès aux diverses variables (en Algol 60, par la procédure "écrire") et qu'un exécutant intelligent d'un algorithme pourrait éviter les calculs dont on ne demande pas la communication à l'extérieur.

Du point de vue théorique, les fonctions récursivement calculables sont définies à partir des trois applications élémentaires suivantes : constante 0, successeur d'un entier, sélection d'un "argument" dans un n-uple (c'est-à-dire projection dans un produit cartésien d'ensemble). Montrer qu'un ordinateur ne recouvre que cette classe de fonctions se ramène à montrer que les instructions machine apparemment plus puissantes appartiennent à cette classe. L'intérêt d'utiliser ces instructions est à la fois économique car le nombre de pas d'un algorithme en est diminué et à la fois méthodologique car les instructions sont plus proches des outils que nous, humains, sommes capables de manier.

Remarque 4 : Notion de processus. Rôle d'un système d'exploitation

Le calcul de la valeur de l'application définie par un algorithme, pour une valeur de l'ensemble de départ, se fait séquentiellement, dans l'ordre des compositions successives. Chaque étape de ce calcul peut être caractérisée par un couple < fonctions restant à appliquer, résultats des applications précédentes > qui est l'ensemble des informations nécessaires et suffisantes qu'il faut conserver pour poursuivre le calcul. C'est précisément ce couple qu'on baptise habituellement "état du processus". On peut donc dire que le processus est le calcul lui-même, tel que le mécanisme de définition d'un algorithme le suggère.

En pratique, la notion d'état d'un processus s'est concrétisée lorsque le problème de la sauvegarde des informations nécessaires à la poursuite du calcul s'est posé. Cette sauvegarde concerna tout d'abord le compteur ordinal et les registres lors d'interruptions de fin d'entrée-sortie, puis la zone mémoire de travail lors du fonctionnement en alternance ("swapping"), puis les tables de correspondance mémoire virtuelle - mémoire physique pour la mémoire virtuelle, et enfin l'état simulé d'un ordinateur et de ses périphériques pour les machines virtuelles. C'est aussi l'état du processus qu'on a imaginé de sauvegarder lors d'un "point de reprise" ("check-point") pour reprendre le travail courant en cas d'incident.

Ces divers cas sous-entendent que le système fait quelque chose en plus des processus demandés par les utilisateurs. En fait, il les inclut dans un processus plus vaste : par exemple, sous OS-360 les programmes utilisateurs sont appelés comme des sous-programmes et devraient revenir au système en fin d'exécution par un retour de sous-programme.

On voit donc que le système s'intéresse aux "combinaisons" d'algorithmes sur un espace en bijection avec l'espace produit de ceux de ces algorithmes, plutôt qu'à un seul algorithme.

De plus, pour réaliser ces combinaisons, il faut introduire des fonctions réalisant ces bijections, qui demeurent mathématiquement "triviales". Citons par exemple : le rangement des registres dans une zone de sauvegarde, le transfert de l'espace de travail d'un programme d'un disque vers la mémoire centrale, l'opération de remplacement de pages de programme.

Ces opérations, en tant que bijections, ne posent pas de problèmes théoriques, mais des problèmes pratiques,lorsqu'on veut les réaliser d'une manière économique ou élégante. Ceci explique pourquoi la formalisation d'une unité centrale sous forme d'algorithme, d'automate, ou de machine de Türing, n'a pas résolu les problèmes des systèmes informatiques.

Remarque 5 : Difficulté majeure des systèmes

Une autre difficulté que rencontre un système informatique est que ni lui-même, ni les programmes des utilisateurs, ne sont en fait les algorithmes souhaités. D'une part ils ne donnent pas toujours le résultat attendu, d'autre part ils n'assurent même pas que le nombre d'étapes à réaliser est fini.

Les actions visant à juguler ces défauts consistent soit à imposer un nombre limite d'étapes, par le truchement du temps qu'une minuterie décomptera, soit à vérifier que les applications reçoivent bien des valeurs dans leur domaine de définition et, pour celles que l'on invente, qu'elles restituent bien une valeur dans le domaine "atteignable".

Ce type de vérification s'est d'abord fait par des mécanismes matériels sur les instructions machine (déroutement pour code d'instruction inexistant, dépassement de capacité, division par 0), pour s'étendre au logiciel (racine carrée d'un nombre négatif, écriture sur un fichier "ouvert" en lecture). Néanmoins, ces vérifications ne sont pas faites systématiquement, faute de mécanismes adéquats. Notons cependant que les langages dans lesquels le type des paramètres est vérifié lors de chaque appel répondent à ce besoin, surtout lorsque les types peuvent être définis par l'utilisateur et concernent le domaine de définition des valeurs (cf. Pascal [84]).

Les garde-fous sur les domaines des applications intervenant dans un processus permettent d'éviter que des informations appartenant à d'autres processus soient détruites. Cela ne résoud pas le problème concernant la validité des résultats qui en pratique est abordé par des vérifications supplémentaires de résultats intermédiaires, c'est-à-dire finalement par des "assertions" qui devraient être vraies si l'algorithme est celui qu'on désire.

La recherche en "preuves de programmes" essaie d'automatiser les preuves de concordance entre l'algorithme décrit et les assertions suggérées par le programmeur.

Insistons encore sur le fait que le système ne connaît pas le déroulement futur des processus utilisateurs qu'il doit mener à bonne fin. Les "combinaisons" qu'il réalisera ne pourront donc pas dépendre de propriétés non exprimées explicitement.

Remarque 6 : Synchronisation des processus

Nous avons évoqué les "combinaisons" d'algorithmes à propos de l'action du système d'exploitation ; approfondissons cette notion.

Lorsque plusieurs algorithmes (en nombre fini) opèrent sur des ensembles différents - en supposant que les ensembles de départ et d'arrivée des applications intervenant dans un algorithme sont tous confondus - c'est-à dire sur des "noms" de variables différents, on peut facilement définir une extension commune de ces algorithmes "composants" qui les réalise tous sur le produit cartésien de leurs ensembles. En effet, tout algorithme sur cet espace produit vérifiant les contraintes suivantes :

. chaque application intervenant dans sa définition se projette sur chaque espace "facteur" soit suivant l'identité, soit suivant l'application qui devrait intervenir dans l'algorithme correspondant, compte tenu des applications précédemment faites,

. il n'y a qu'un nombre fini d'applications identité,

fera se dérouler sur chaque espace facteur l'algorithme composant correspondant, avec insertion d'un nombre fini d'applications identité.

En pratique, le cas particulier où les applications de l'algorithme extension n'ont qu'une seule projection différente de l'identité, cas que nous appellerons "multiplexage", est celui où une seule exécution peut se faire à la fois : traitement des interruptions, des symbionts, de la multiprogrammation en mono-processeur.

D'un autre côté, chacune des étapes d'un algorithme peut être considérée comme un "algorithme" infini ("pseudo-algorithme") où elle serait répétée sans fin. Alors l'algorithme lui-même peut s'interpréter comme une combinaison, un multiplexage même, de ces "pseudo-algorithmes", où l'ordre des compositions d'applications est strictement défini et leur nombre reste fini. Il est intéressant de noter que les langages de programmation, tel qu'Algol 68, essaient de s'affranchir de cette rigueur : il est possible de spécifier que certaines séquences d'actions peuvent s'exécuter "en parallèle" c'est-à-dire finalement que toutes les combinaisons de ces séquences, considérées comme des algorithmes, sont valides.

L'ordre des compositions assurant que les étapes qui utilisent des variables communes "coopèrent" de façon satisfaisante, est un élément essentiel de la définition d'un algorithme. C'est pourquoi chaque combinaison d'algorithmes (ou de "pseudo-algorithmes") utilisant des variables communes transforme l'effet de ces algorithmes composants. On peut donc être amené à rechercher la "classe" de combinaisons qui transforment des algorithmes composants et vérifient certaines propriétés caractéristiques ou prédicats.

Par exemple, la relation "producteur - consommateur" qui a conduit Dijkstra à définir les "sémaphores" [32] peut se représenter par la combinaison des deux algorithmes suivants :

i) "producteur" qui, entre autres actions, va faire croître une variable entière k, nombre d'unités produites, de 0 à N,

ii) "consommateur" qui, entre autres actions, va faire croître une variable entière l, nombres d'unités consommées, de 0 à N,

qui doit respecter les deux conditions suivantes :

- on ne peut consommer plus qu'on a produit ($\Leftrightarrow l \leqslant k$)
- la capacité de stockage est limitée à n ($\Leftrightarrow k \leqslant l+n$)

Une méthode générale pour assurer que les propriétés voulues restent vraies est d'assujettir l'exécution de chaque application à la vérification préalable des propriétés transformées par celle-ci. En fait, on limite cette vérification aux applications risquant d'affecter certaines propriétés, c'est-à-dire dans l'espace d'arrivée desquelles une, au moins, des variables de ces propriétés intervient.

Par exemple, dans l'algorithme "producteur" cité précédemment, il faudra remplacer "k := k + 1 ;" par

"vérif : si (k + 1 ⩽ 1 + n) alors k := k + 1 sinon allera verif ;". En effet, si les conditions (1 ⩽ k) et (k ⩽ 1 + n) sont vérifiées avant cette application, nous aurons 1 ⩽ k après, tandis que la seconde ne restera valide que si k + 1 ⩽ 1 + n.

Notons qu'aucune action ne peut être intercallée entre la vérification (suivant le si) et l'action (suivant le alors) sous peine d'invalider les conditions. Ce problème "d'indivisibilité" de certaines actions peut être résolu à partir de l'indivisibilité du test et de l'écriture d'une seule variable booléenne, permise, en pratique, par l'instruction "test and set" (IBM 360, Univac 1108, GE 645, IRIS 80, etc ..).

Cette manière de faire consiste à reporter dans les algorithmes composants des contraintes sur leurs combinaisons. Afin d'éviter que les attentes ainsi introduites (cf."allera verif") nuisent à la combinaison, Dijkstra a imaginé des actions dans les algorithmes composants qui auraient un pouvoir sur les combinaisons réalisables. Sur un "sémaphore" s, variable entière, deux actions sont définies :

p(s) : qui vérifie une condition et dans le cas où elle n'est pas vérifiée "indique" que toutes les combinaisons incluant l'action sont invalides. On dit alors que le processus (exécution de l'algorithme) est bloqué.

v(s) : qui rend éventuellement valide une condition et qui dans ce cas "indique" que des combinaisons jugées invalides auparavant sont maintenant valides (déblocage ou réveil d'un processus).

Précisons le fonctionnement de p(s) et de v(s) : les actions associées sont respectivement s := s - 1 et s := s + 1 et la condition voulue est s ⩾ 0.

On peut donc exprimer les actions p et v sous forme de "pseudo" procédure Algol :

```
    p(s) :
début
    s := s - 1 ;
    si s < 0 alors le processus
            se bloque
fin ;
```

```
    v(s) :
début
    s := s + 1 ;
    si s ⩽ 0 alors réveiller un
            processus bloqué sur s
end ;
```

On remarque que l'action du p est faite de toute façon car elle permet d'une part de ne pas revenir sur le test et d'autre part de compter les processus bloqués (si $s < 0$ il y en a $-s$).

Dans ces conditions, l'exemple précédent peut être traité par des sémaphores $s_1 = 1 + n - k$, nombre de places restant libres pour le stockage, et $s_2 = k - 1$, nombre d'unités produites mais non encore consommées. Alors l'action de produire "k := k + 1 ;" transformée en

"verif : si(k + 1 ⩽ 1 + n) alors k := k + 1 sinon allera verif ;" peut s'écrire à l'aide des sémaphores s_1 et s_2 : "$p(s_1)$; $v(s_2)$;" car l'action $v(s_2)$ ne fait que répercuter sur s_2 l'incrémentation faite sur k. De même l'action "l := l -1 ;" de consommation se trouverait exprimée par "$p(s_2)$; $v(s_1)$;".

Si l'action de produire nécessite plusieurs actions séquentielles, on peut différencier le nombre d'unités qu'on a commencé à produire k_1, de celles qu'on a terminé de produire k_2. L'action de produire peut s'écrire :

$k_1 := k_1 + 1$;
[actions de production] ;*
$k_2 := k_2 + 1$;

---

* [ ] remplace une séquence d'instructions arbitraires sauf allera vers "l'extérieur".

En dédoublant de la même façon 1, le nombre d'unités consommées, les relations de cohérence deviennent plus précises :

. on ne peut commencer à consommer plus qu'on a fini de produire ($l_1 < k_2$) ;

. on ne peut commencer à produire plus de n unités (capacité de stockage) qu'on a fini de consommer ($k_1 < l_2 + n$). En procédant comme précédemment à l'aide des sémaphores $s_1 = l_2 + n - k_1$ et $s_2 = k_2 - l_1$, les actions de production et de consommation deviennent donc respectivement "$p(s_1)$; [actions de production] ; $v(s_2)$;" et "$p(s_2)$; [actions de consommation] ; $v(s_2)$;". Alors que le nombre des variables a augmenté, l'expression à l'aide des sémaphores est restée la même, ceux-ci sont donc aussi adaptés aux actions non indivisibles.

Il me semble que l'expression précise des conditions de cohérence permet de formuler la solution d'un problème avec les fonctions p et v sur sémaphores plus naturellement et exactement que dans la pratique courante où l'intuition et l'expérience acquise jouent le plus grand rôle.

La méthode de vérification dynamique des conditions de cohérence empêche le déroulement de combinaisons invalides, mais peut conduire à des impasses : tous les processus peuvent se trouver bloqués, c'est-à-dire que la combinaison commencée ne peut plus trouver de "suite" satisfaisant aux relations. La prévention de tels cas, dits "d'étreinte mortelle" ("deadlock") a pu se faire dans certains cas très restrictifs (cf. Haberman [38]). En pratique, les systèmes n'effectuant pas cette prévention doivent détecter l'étreinte mortelle. Généralement c'est l'opérateur qui doit s'en apercevoir et qui élimine alors l'un des processus !

On voit donc que la souplesse permise dans l'expression des algorithmes par les actions p et v, ou plus généralement par les synchronisations de processus, n'a pas trouvé sa contrepartie dans la sécurité, autant pour prévenir les étreintes mortelles, que pour traiter des malfonctions dans un algorithme composant.

Notons que la manière de décomposer un algorithme en algorithmes qui se synchronisent n'est pas unique. En appendice 2, nous montrons un exemple inspiré de fructueuses discussions avec Anceau [2], où la même action est exprimée soit par un p, soit par un v, suivant la représentation adoptée.

3.1.2. Représentation du temps dans un simulateur

Supposons que le système à simuler peut être représenté ainsi à l'instant $t(t \geq 0)$: un vecteur de variables ayant une valeur $V(t)$, un ensemble de fonctions $\{f_i\}$ susceptibles d'être appliquées aux variables.

Chacune de ces fonctions définit un nouveau vecteur de variables et un nouvel instant $t' > t$ auquel cette transformation serait effective : $V(t')$.

Une méthode applicable de façon générale répètera alternativement les deux étapes suivantes :

i) trouver l'évènement significatif le plus proche dans le futur
ii) simuler l'évolution du système jusqu'à cet instant

La recherche de l'évènement significatif le plus proche peut impliquer des essais de simulation inutiles : par exemple, si l'on simule une unité centrale et un canal, on peut trouver, en considérant seulement l'U.C. que le prochain évènement sera un déroutement pour défaut de page du programme à t', puis en considérant le canal que celui-ci demande une interruption de l'unité centrale à $t'' < t'$, en fin de transfert.

La détermination de t' a été inutile. Inversement, en considérant d'abord le canal on peut trouver une fin de transfert à t', puis en considérant l'U.C. on trouve une opération arrêtant l'activité du canal (HIO) à $t'' < t'$. Ces exemples montrent qu'il peut être impossible de déterminer un ordre de considération des évènements qui éviterait cet effet.

Cette progression est la méthode utilisée dans les langages de simulation comme SIMSCRIPT [47].

Dans le cas particulier, où $t'-t$ est multiple d'un intervalle minimum Δt, on peut simuler l'évolution du système suivant les "tops" d'horloge à intervalle Δt. On évite ainsi des prédictions inutiles, mais il se peut qu'on considère des actions qui progressent sans évènements significatifs.

La difficulté majeure provient de l'ordre de grandeur comparé de l'intervalle de base et de celui pendant lequel le phénomène à étudier est significatif. Cette méthode est utilisée pour la simulation des systèmes logiques ou les systèmes à changements continus (cf.DYNAMO [68]).

Nous allons montrer que la décomposition en algorithmes (ou processus) de l'activité du système permet de mieux rechercher les effets locaux et d'en tirer parti en rendant la simulation plus efficace.

Supposons que chaque étape (ou application) d'un algorithme vérifie l'hypothèse suivante :

H1 - a) - elle est indivisible dans le temps.

b) - elle n'utilise les variables communes à plusieurs processus, appelées variables globales que dans les conditions suivantes :

- au début de son action, elle peut en lire certaines
- à la fin de son action, elle peut en modifier certaines

c) - la durée de son action est strictement positive.

Cette hypothèse confère un caractère local au déroulement d'une étape, qui une fois commencée n'a plus de relation avec les autres algorithmes.

On a formulé l'idée de la durée d'une action, mais les évènements début et fin de l'action et la définition des instants d'occurence auraient aussi bien convenu.

Supposons que chaque algorithme soit muni d'une horloge marquant l'heure universelle dont l'avancement obéit à la règle suivante :

H2 - l'avancement de l'horloge est fait, en fin d'étape avant modification des variables globales.

Alors un mécanisme de multiplexage des algorithmes qui capte ces demandes d'avancement et respecte la règle suivante :

H3 - lors d'une demande d'avancement remettre l'horloge du processus appelant à l'heure, et donner le contrôle à l'un des processus (il peut y en avoir plusieurs) dont l'horloge est la moins avancée,
assurera que tous les processus utilisent les bonnes valeurs temporelles des variables globales, sauf dans le cas où plusieurs processus utilisent simultanémei les mêmes variables globales dans la réalité, ce qui, en général, est ambigü

Afin de démontrer ceci, définissons la t-valeur d'une variables globale comme sa valeur à l'instant t dans la réalité.

Supposons que l'on donne le contrôle à un processus qui a demandé que son horloge soit mise à t. Les variables globales qu'il va lire ont leur bonne valeur si et seulement si toutes les fins d'actions strictement antérieures ont réalisé leurs modifications et aucune fin d'action strictement postérieure ne l'a fait.
Montrons par l'absurde :

S'il existe t' < t, tel qu'une variable globale devant être modifiée à t' ne l'a pas été, d'après H2 cela signifie que le processus correspondant a demandé sa mise à jour à t' et ne l'a pas obtenu. Comme t' < t, il est absurde que le processus courant ait obtenu le contrôle à t avant ce dernier, d'après H3.

De même, s'il existe t" > t, tel qu'une variable devant être modifiée à t" l'a été, d'après H2 cela signifie que le processus correspondant a demandé sa mise à jour à t" et l'a obtenue. Comme t" > t, il est absurde que ce dernier ait obtenu le contrôle avant le processus courant à t, d'après H3. Ainsi, l'ordre chronologique est respecté.

En ce qui concerne les actions simultanées sur les mêmes variables globales, l'hypothèse H3 laisse indéterminé l'ordre d'occurence des évènements.

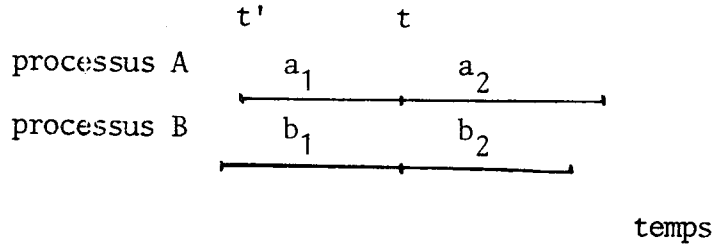


figure 3.1.

Considérons par exemple, deux processus A et B qui en sont à l'instant t. Supposons que la dernière action du processus A, a_1, modifie une variable globale V, et que la prochaine action du processus B, b_2, la lise. En appliquant H3, suivant que l'on simule d'abord a_2 puis b_2 ou d'abord b_2 puis a_2, à cause de la règle H2, le processus B va utiliser soit la t-valeur de V soit la t'-valeur.

Dans la pratique ces coïncidences peuvent être évitées par des règles du genre "une écriture précède toujours une lecture", mais ceci est insuffisant pour expliquer ce qu'il va se passer si plusieurs écritures arrivent simultanément.

De telles règles introduites dans le simulateur ne sont pas compatibles avec les précédentes : en effet si l'on introduit une variable V' dans l'exemple précédent, modifiée par b_1 et lue par a_2, on voit qu'il est impossible de simuler correctement l'effet recherché (écriture avant lecture) à l'aide des règles de simulation des processus.

Concluons en disant que ces coïncidences n'ont pas besoin d'être empêchées, si d'autres raisons garantissent le bon fonctionnement (cela ne peut arriver que pour des lectures, par exemple), mais qu'en général, il vaut mieux systématiquement qu'elles ne se produisent pas compte tenu des effets désastreux qu'elles pourraient entraîner.

Les primitives de mutuelle exclusion dont nous avons parlé dans le paragraphe précédent, sont adaptées à cette expression, dans le cadre des simulateurs. On peut donc ajouter l'hypothèse H4 suivante .

H4 - Lorsque des processus utilisent des variables globales, ils en demandent au préalable l'accès exclusif.

Lorsque ces accès sont interdits, le processus se trouve "hors du temps", et sera réveillé par la libération qu'un autre processus aura faite à l'heure correspondant à cette action. Pour ce faire, la remise à l'heure de l'horloge devra précéder chaque opération "v".

Les hypothèses H1 à H4 garantissent le bon usage temporel des variables. On peut employer ces règles, comme des critères nécessaires et suffisants pour agglomérer plusieurs actions élémentaires en une seule.

La condition nécessaire et suffisante est que "l'intérieur" de la nouvelle action ne comporte aucun accès ni aucune modification de variables globales.

Ainsi par exemple, on peut regrouper en une seule action :
une action lisant une variable globale V et n'en modifiant aucune,
suivie d'une action sur variables locales,
suivie d'une action ne lisant que des variables locales puis modifiant la variable globale V'.

Pour effectuer les autres regroupements , il faut avoir des preuves logiques, soit du fait qu'on peut repousser l'écriture d'une variable globale parce qu'aucun autre processus ne va s'en servir pendant l'intervalle, soit qu'on peut avancer la lecture d'une variable globale parce qu'aucun autre processus ne l'a modifié dans l'intervalle.En particulier, lorsqu'un processus en multiplexe plusieurs autres, on peut intervertir les actions correspondantes pourvu que l'effet sur les variables globales soit invariant.

Par exemple, si le traitement d'interruption qu'effectue une unité centra ne perturbe pas la "tâche" interrompue, on peut reporter le traitement d'interruption à la fin logique de la tâche, c'est-à-dire au premier évènement significatif du point de vue des variables globales.

Notre simulation de CP a utilisé ce phénomène en montrant que le déroulement logique était inchangé, et que les commutations ainsi économisées accélèrent grandement les performances du simulateur.

3.1.3. Structure d'un simulateur

D'après ce qui précède, le simulateur chargé de l'avancement de l'horloge des processus doit aussi gérer les demandes d'action exclusive.

Les structures de données qu'il peut utiliser seront :

- une liste des processus non bloqués sur demande d'action exclusive, ordonnée suivant les temps croissants
- pour chaque sémaphore, une liste des processus en attente passive.

Chaque élément des listes sera en fait un groupe d'informations contenant l'horloge, le contexte (qui permet de poursuivre le processus) et divers compteurs pouvant intéresser l'expérimentation.

Les procédures élémentaires peuvent se réduire à trois :

avancer(Δt) : demande par un processus d'avancer son horloge de Δt unité(s) de temps

p(s) et v(s) : les fonctions sur sémaphores introduites par Dijskstra

Si l'on désire la création ou la destruction de processus ou de sémaphores pendant la simulation, des fonctions correspondantes s'ajouteront aux précédentes. L'effort nécessaire pour réaliser ces fonctions est minime. Il ne faut pas craindre de l'entreprendre si les expériences de simulation doivent être nombreuses et si les langages de simulation existants ont des défauts en ce qui concerne les expériences envisagées.

Parmi ceux-ci notons : la notion de processus qui s'attache à la tâche programme plutôt qu'à la ressource physique, et l'absence d'un langage algorithmique sous-jacent dans GPSS 360 ; l'absence de notion de processus dans SIMSCRIPT II ; l'absence des mécanismes d'accès exclusifs dans SIMULA 67, bien qu'il soit facile de les fabriquer.

Teichroew et Lubin [79] ont comparé intelligemment les langages disponibles en 1966. Certaines caractéristiques ont été améliorées depuis lors, et de nouveaux langages sont apparus (Simula 67), qui rendent caduques leurs comparaisons. On peut cependant retenir le mal qu'ont les langages de simulation à suivre l'évolution des langages généraux.

Dans les sections suivantes nous montrerons quels dispositifs et en particulier quels langages, peuvent aider à exprimer le modèle ou à conduire les expériences de simulation.

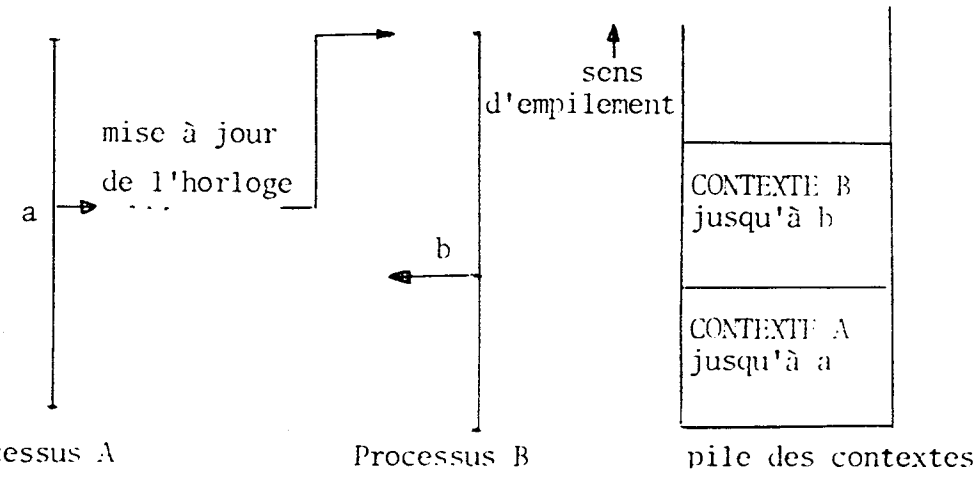
3.2. EXPRESSION ALGORITHMIQUE DU MODELE

Les langages de simulation spécialisés n'ont pu suivre l'évolution des langages généraux en ce qui concerne soit les épurations de syntaxe, soit l'unification de concepts (instruction, procédure et valeur).

Par exemple, GPSS 360 [78], successeur de GPSS I, II et III, ne permet pas l'usage de fonctions qui ne sont pas sous forme tabulée, impose celui des variables et paramètres numérotés, ne permet pas d'expression arithmétique assez complexe. Simscript II [47] au niveau de Fortran le permet mais n'a pas les structures conditionnelles comme while ou case...of . Simula 67, [82] sur-ensemble d'Algol 60, permet l'usage de macro-notation dans les procédures, unifie le concept de blocs de données et de procédures (instances de classe) mais à un prix qui semble exhorbitant. Par contre, la notation "pointée" qui est différente de la notation de procédure (objet. propriété signifie propriété de objet), l'affectation de pointeur spéciale (: - au lieu de :=), l'usage implicite de l'environnement dans les noms de variables rendent les programmes peu lisibles.

On doit cependant reconnaître que les langages généraux n'ont pas non plus une construction unique pour "propriété de objet" suivant qu'une procédure, que le champ d'un bloc de données pointée, qu'un tableau matérialise cette fonction. Ceci est un obstacle important à la construction par raffinement successifs d'algorithme ("top-down") car on est obligé d'adopter prématurément une représentation des données, ce qui rend difficile un changement ultérieur.

Un autre aspect des langages généraux qui ne convient pas à la simulation est l'empilage des environnements ou contextes. En effet l'acquisition dynamique de mémoire ou la récursivité des procédures est habituellement traitée grâce à un empilage des contextes. On atteint ainsi une structure dynamique et l'on satisfait aux règles sur la portée des identificateurs. Mais si l'on doit simuler deux processus A et B, tel qu'au milieu de A on demande un avancement d 'horloge qui conduise à entamer le processus B, alors on se retrouve dans la situation suivante :



On a empilé le contexte de B sur celui de A. Comment pourra-t-on alors sans détruire le contexte de B, revenir à celui de A.

Notons que les systèmes d'exploitation à multiprogrammation, ne peuvent , eux non plus, se servir d'une pile unique cablée pour assurer le multiplexage des processus.

La solution idéale consiste en plusieurs piles indépendantes pour chaque processus. Néanmoins, la croissance de l'espace commun, racine commune de ces piles poserait des problèmes.

Simula autorise ces créations de piles, associées aux créations de processus, mais les dépendances qu'il permet peuvent provoquer des actions presque incompréhensibles, y compris aux auteurs du langage et des compilateurs. Afin d'utiliser PL/1 comme langage de simulation, Jones [45] a été obligé de réaliser des primitives de duplication de piles de contexte, parmi d'autres extensions.

Dans quelques cas simples, le problème se trouve éludé :

i) - pas d'allocation dynamique, donc pas de pile. Ainsi le simulateur GASP [48], constitué de routines incorporées à Fortranqui alloue statiquement l'espace, n'a pas ce problème à résoudre.

ii) - contexte vide lors de l'appel de l'horloge. Ceci ne peut survenir que si l'exécution d'une action élémentaire d'un processus est conçue comme un appel de procédure, dont le retour correspond soit à l'avancement d'horloge, soit à une attente sur un sémaphore.

La règle de simulation obligeant ces actions à précéder les modifications de variables globales (cf. 3.1.3) implique que des copies de ces variables devront être maintenues dans un espace commun. C'est cette solution que nous avons employée avec le langage AED, dérivé d'Algol 60 (cf. Ross [72]).

3.3. MESURES ET CONTROLE DES EXPERIENCES

Mentionnons les traits essentiels des instruments de mesures dans un simulateur.

a)- Compromis entre la séparation des mesures et des algorithmes, et la sélectivi des mesures

Les diverses solutions évoluent entre deux extrêmes : mesures définies de façon standard sans besoin d'expression, qui réalisent donc une séparation parfaite mais sans sélectivité, et mesures définies par des instructions dans le modèle qui réalisent une sélectivité parfaite mais sans séparation.

En fait la sélectivité se résume à faire une collection de procédures de mesure, comptage, moyenne, variance, histogramme, corrélations etc... que l'on peut enrichir soi-même et à choisir lesquelles s'appliqueront aux diverses variables.

Par macro-notation cette sélectivité peut être accomplie et la séparation entre mesures et algorithme conservée. Ceci peut être fait dans certains langages comme Algol 68 ou Simula en faisant dépendre l'action entreprise, ici les mesures, du "type" de variable considéré.

Par exemple, considérons une file d'attente f sur laquelle on veur définir l'insertion d'un élément e (insérer (f,e)) et la sortie d'un élément (retirer (f,e) où e est l'élément choisi pour sortir de la file).

Sur cette file on veut pouvoir opérer les mesures suivantes (cf. Mac Dougall [58]) : valeur moyenne de la longueur de la file,
valeur maximum de la longueur de la file,
temps moyen d'attente dans la file,
temps maximum d'attente dans la file,
si et seulement si les valeurs des indicateurs booléens respectifs l(f), lm(f), t(f), tm(f), sont vraies.
Alors en définissant conditionnellement les données suivantes :

- pour chaque élément
  if t(f) or tm(f) then T entrée(e)
  (instant de la dernière entrée de e dans la file f)
- pour la file f
  if t(f) or tm(f) then T dernier(f) (dernier instant d'opération sur f).
  if t(f) then S attente(f) (somme des temps d'attente)
  if tm(f) then M attente(f) (attente maximum)

```
if l(f) or lm(f) then elements(f)    (nombre d'éléments dans la file)
if l(f) then appels(f)    (nombre d'appels à la procédure retirer)
if l(f) then Selements(f)    (intégrale dans le temps du nombre d'éléments)
if lm(f) then Melements(f)    (nombre maximum d'éléments)
```

On peut opérer sélectivement les mesures en mettant à jour les informations suivantes :

- lors d'insérer (f,e) à l'instant T :

```
if l(f) then (Sélements(f)plus (T-Tdernier(f))*elements(f) ; Tdernier(f):=T) ;
if l(f) or lm(f) then elements(f)plus 1 ;
if lm(f) then Melements(f):=max (Melements(f), elements(f)) ;
if t(f) or tm(f) then Tentrée(e):=T ;
```

- lors de retirer (f,e) à l'instant T (lorsque e est défini):

```
if l(f) then (Selements(f)plus (T-Tdernier(f))* elements(f)) ;Tdernier(f):=T) ;
if l(f) or lm(f) then elements(f) minus 1 ;
if t(f) then (appels(f)plus 1 ; Sattente(f)plus (T-Tentree(e))) ;
if tm(f) then Mattente(f):= max(Mattente(f), T-Tentree(e)) ;
```

à un instant T d'appel de ces procédures, nous obtiendrons les mesures voulues par les relations :

nombre moyen d'éléments dans la file	: $\frac{\text{Sélements}(f)}{T}$
attente moyenne d'un élément	: $\frac{\text{Sattente}(f)}{\text{appels}}$
nombre maximum d'éléments	: Mélements(f)
attente maximum d'un élément	: Mattente(f)

Les conditions précédentes prises en compte à la compilation permettent la sélectivité , sans nuire aux performances comme le ferait un examen à l'exécution.

b) - définition dynamique de mesures en cours d'expérience

Le contrôle conversationnel d'une expérience peut être utile à plus d'un titre : au cours de la mise au point, il permet de limiter le coût d'expériences faussées par des erreurs dans le modèle. Au cours d'expériences normales, il permet de définir les fréquences de prises de mesure, au vu des résultats obtenus. Au cours d'expériences où des phénomènes inattendus se produisent, le contrôle conversationnel permet de prendre des mesures détaillées des seuls instants où le phénomène apparaît.

La réalisation de mesures sous cette forme peut se faire de manière élégante ; alors que la prise de mesures se fait au cours de l'exécution du modèle, leur extraction et affichage se fait par un processus "espion" immatériel, c'est-à-dire ne consommant aucune ressource. Ce processus est activé soit périodiquement, soit par des évènements spéciaux. Pour que l'intéraction soit véritable il faut un langage de commande qui permette de définir de nouvelles interrogations suivant les expériences envisagées.

Le processus espion comporte donc un interpréteur de ce langage de commande dont la syntaxe par ailleurs peut être fort simple :

<commande> ::= liste d'<item> séparés par des blancs

<item > ::= chaîne de <caractère> |<nombre>

Outre des commandes d'affichage des mesures prises jusqu'alors ou des diverses structures de données utilisées, on peut commander la prise de traces par la fixation d'un niveau, qui peut rendre effectif des instructions conditionnelles du type

if niveau(trace) ≥ n then écrire("information") ;

qui auront été prévues dans le modèle. Signalons des particularités du système AED qui se révèlent très utiles dans la spécification conversationnelle de commande. Au lieu d'une procédure de lecture de ligne, le système met à la disposition des usagers une procédure de lecture d'"item" reconnus à la compilation, en particulier, nombre ou identificateur. Il n'est pas difficile alors d'écrire un langage de commande.

De plus dans le système CMS, on peut écrire une ligne entière, ce qui permet de spécifier plusieurs paramètres à l'avance. Le système AED permet d'inclure un message de sortie avant chaque paramètre qui n'est pas spécifié à l'avance. Ce message sert à préciser la nature et la forme du paramètre attendu. On obtient ainsi deux formes de conversation ; l'une concise convenant à un utilisateur averti et l'autre prolixe convenant à un utilisateur débutant ou à l'auteur du programme après quelques mois d'inutilisation. Par exemple :

```
COMMANDE  : i
INTERVALLE :500
COMMANDE  : u
NUMERO D'UTILISATEUR : 15
COMMANDE  : f
```

sera équivalent à

```
(COMMANDE:) i 500 u 15 f .
```

Notons que le danger d'oubli d'un paramètre lors d'une expression concise est à craindre, aussi est-il préférable que le programme renvoie certains paramètres qui ont été enregistrés afin d'alerter l'utilisateur au cas où une erreur se serait produite.

En fait compte tenu de la généralité du problème de la mise au point, il apparaît essentiel de concevoir un outil de ce genre lorsqu'on veut fabriquer une application importante.

Signalons enfin que des garde-fous peuvent être prévus dans le modèle, sous la forme de vérifications de cohérence. Celles-ci permettent d'identifier de manière interne des malfonctions puis de réveiller le processus espion qui analyse plus complètement l'état du modèle.

c)- fabrication de fichiers de mesures

Alors que l'examen conversationnel s'apparente au pilotage, il peut être nécessaire de faire, après expérience, des traitements mathématiques plus complets (régression typologie, analyse factorielle). Aussi importe-t-il de conserver les résultats sous forme de fichiers pour ne pas avoir à les retranscrire manuellement. Naturellement la gestion de tels fichiers est délicate,car il faut identifier les expériences, écarter celles qui ont été faussées. Dans les expériences de simulation auxquelles j'ai participé ([11], [59]), les investissements liés aux prises de mesures ont toujours semblé largement rentables. C'est donc un outil de première nécessité.

3.4. INITIALISATION DES EXPERIENCES

Dès que le nombre de paramètres d'un modèle devient important, il devient fastidieux de les spécifier, surtout de manière conversationnelle (par opposition à des cartes "données"). Dans la pratique on effectue une partition entre des paramètres initialisés par programme, donc relativement fixes et ceux que l'on peut (et doit) redéfinir à chaque expérience.

Lorsque ces paramètres restent nombreux, il peut être souhaitable de leur donner une valeur par défaut, définissant un point de fonctionnement à partir duquel tous les autres rayonneront.

Certains de ces paramètres peuvent n'être pas numériques, comme des algorithmes de décision par exemple. Le langage doit pour les admettre aisément avoir la notion de référence ou pointeur vers une procédure, sinon il faut, sous la même procédure, procéder à un aiguillage (cf. points d'entrée multiples de PL/1). Notons que d'autres, d'apparence numérique recouvrent des spécifications algorithmiques. Par exemple, pour passer de 1 à 2 tambours ou de 1 à 2 unités centrales, il faut spécifier comment les organes multiples coopèreront. De plus la spécification des paramètres peut être hiérarchique : Du choix entre les paramètres A et B peut découler le choix des paramètres a_1, a_2 ou b_1, b_2, b_3 qui n'ont de sens que dans 1 seul des choix initiaux.

Un langage de commande assez comparable à celui concernant la prise de mesures peut répondre à ce besoin de spécification. Il doit permettre notamment de renseigner l'utilisateur sur :

- le sens des paramètres que l'on veut changer
- leur valeur par défaut
- la dépendance hiérarchique des paramètres

Les modes de conversation concis et prolixes conviennent aussi parfaitement à ce cas. En fait chaque partie paramétrisable du simulateur doit trouver sa contre-partie dans la documentation conversationnelle et son initialisation dynamique.

La gestion dynamique d'espace par le simulateur est un atout supplémentaire pour ce faire. Notons qu'il vaut mieux alors appeler un nouvel exemplaire du simulateur pour chaque expérience plutôt que restituer tous les blocs occupés en fin d'expérience, quand cela est possible.

3.5. SPECIFICATION DES DONNEES D'ENTREE

Les données d'entrée, par exemple les tâches d'utilisateurs dans un système informatique, peuvent être définies de manière diverses. Elles peuvent provenir de "traces" de données réelles enregistrées sur fichier ou interprétées, de procédures engendrant des nombres pseudo-aléatoires dans le cas de définition probabiliste, ou d'un algorithme quelconque.

Il est très souvent nécessaire d'effectuer des transformations à partir des données "brutes" ou d'autres algorithmes. La notion de générateur de données d'entrée permet à toutes les sources possibles d'être connectées au simulateur.

En fait la relation entre ce générateur et le modèle peut être rendue symétrique :

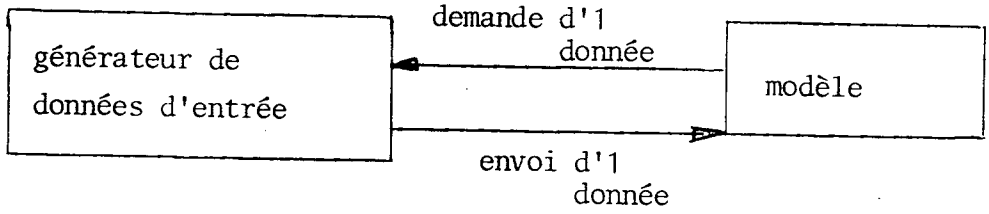


figure 3.3.

en considérant ces deux parties comme des co-routines.
Quand le modèle des données d'entrée est complexe, le générateur peut ressembler à un interpréteur, qui d'après des données synthétiques (nom du fichier de trace ou loi probabiliste) va fabriquer les données que désire le modèle. A l'initialisation, certains de ces paramètres peuvent être spécifiables ce qui conduit à une forme d'expression des entrées proches d'un langage où des itérations et des variables qui en dépendent pourront être exprimées.

Lorsque des variables aléatoires seront utilisées, il est important que ce générateur vérifie la forme effective des données engendrées. En effet, des dépendances entre variables ou des distorsions peuvent apparaître à cause des procédures de génération adoptées, et en particulier lorsque toutes les variables sont issues de la même procédure de génération de variable aléatoire uniforme. Des tests statistiques de moyenne, de variance (X^2), de corrélation pourront ainsi être faits.

CHAPITRE 4

LE MODELE DE SIMULATION DE CP/CMS

4.1. Introduction

Avant de décrire les buts souhaités par les expériences de simulation, décrivons brièvement le système CP/CMS.

4.1.1. Description sommaire de CP-67

CP (Control Program) est un système qui utilise le principe des "machines virtuelles" : un certain nombre de programmes écrits pour des IBM 360 standard (c'est-à-dire autres que le modèle 67) peuvent s'exécuter comme si chacun bénéficiait d'une machine (appelée virtuelle).

La fécondité de ce principe provient de trois faits principaux :

1) tout programme pour système réel sur un IBM 360 constitue un programme pour système virtuel (vaste bibliothèque d'applications) ;

2) CP assure le multiplexage entre systèmes virtuels. Ceux-ci peuvent fonctionner en mono-programmation sans même essayer d'utiliser la simultanéité entre canaux d'entrée-sortie et unité centrale (cf. CMS, 'Cambridge Monitor System'). En effet, la gestion de matériel virtuel risque d'être inefficace si elle n'est pas faite en collaboration avec CP. CP essaiera de prendre avantage des temps d'attente des systèmes virtuels (notamment lors de la frappe d'une ligne à la machine à écrire) pour faire alterner le contrôle donné à chacune d'elles ;

3) les instructions de "travail" (c'est-à-dire non privilégiées) des machines virtuelles sont exécutées directement, sans interprétation par CP, grâce à un dispositif d'adressage spécial au modèle 360/67, dont le principe a été repris dans la série 370.

Les adresses utilisées par le programme des machines virtuelles sont traduites en adresses réelles avant d'être effectivement utilisées par le 67. Cette traduction porte sur des blocs de 4096 octets appelés "pages" et est ignorée par la machine virtuelle. Quand un programme virtuel n'utilise que des pages présentes en mémoire réelle, le processus de traduction devient très rapide (moins de 50 ns) grâce à des registres associatifs et la vitesse d'exécution du programme virtuel approche celle du programme sur une machine sans pagination. CP-67 assure la correspondance entre les pages virtuelles et les pages physiques et gère celles-ci de manière à satisfaire la demande des diverses machines virtuelles (interruption pour page manquante dans le processus de traduction).

Une description plus détaillée de CP est faite dans l'article de Auroux et Hans [4] ; la notion de "virtualisation" est approfondie dans l'article de Goldberg [36].

Les principes de base de CP en font un système peu complexe puisque des programmes comme la gestion de fichiers ou la traduction de langages de haut niveau sont reportés au niveau des systèmes virtuels.

Le souci de son amélioration s'est manifesté très tôt et l'Equipe du Centre Scientifique IBM de Cambridge (Massachusetts, USA), responsable de son développement, comprend un groupe de chercheurs mathématiciens qui participent activement à l'évolution du système.

4.1.2. But des expériences

Le but des expériences que le simulateur devait entreprendre, était d'essayer de prédire l'effet de certains changements (configuration, algorithmes de gestion) sur les performances du système. Plus précisément, l'influence des algorithmes de remplacement (pagination) et celle des méthodes d'ordonnancement semblaient considérables.

Il est important de noter que ces objectifs recquièrent l'introduction de modèles réalistes d'utilisation de la mémoire par les programmes des machines virtuelles.

D'autres simulations sur des systèmes paginés ont souvent utilisé des modèles simplifiés de comportement de programmes où l'identité des pages n'apparaît pas, ce qui interdit d'apprécier des algorithmes de remplacement. C'est le cas du simulateur du système MOS sur Spectra 70/46 de RCA [66] dont la validation a été faite par ajustage de paramètres jusqu'à accord à 5 % près sur le temps de réponse et sur le taux d'activité de l'unité centrale. Les résultats prédictifs de cette simulation ont, d'après leurs auteurs, une incertitude relative de 15 %. C'est aussi le cas du simulateur d'Esope sur 10070 de la CII [12] qui n'a d'ailleurs pas été validé.

De plus, nous voulions vérifier si un simulateur tenant compte de la pagination pouvait fonctionner avec un rapport temps simulé sur temps d'expérience de l'ordre de l'unité (disons de 1/10 à 10). Comme nous le verrons, cet objectif peut être atteint et l'on peut relier le degré de "finesse" de la simulation et le rapport précédent.

4.2. Choix généraux dans le modèle

4.2.1. Structure du modèle

On a choisi de représenter le modèle par un ensemble de processus qui décrivent l'évolution temporelle des ressources dites "actives" (unité centrale, canaux d'entrées-sorties). Ces ressources exécutent des séquences d'actions appelées "tâches", définies indépendamment des autres ressources. Leur activité peut être décrite par un cycle de trois procédures :

1) la procédure de sélection qui choisit une tâche parmi toutes celles qui attendent d'être exécutées par la ressource, ou met celle-ci en attente s'il n'y en a pas ;

2) la procédure d'exécution de la tâche choisie ;

3) la procédure de mise à jour qui crée ou réveille d'autres tâches.

Chacune de ces procédures est une "action élémentaire" au sens de 1.3.2., moyennant certaines hypothèses simplificatrices qui seront énumérées lors de la description du cycle de la ressource correspondante.

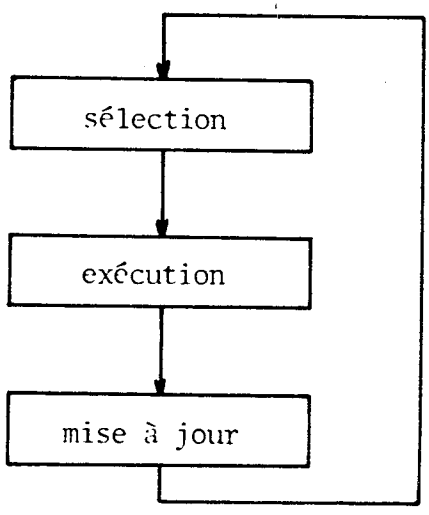


Figure 4.1.

4.2.2. Modèle des tâches

Le travail effectué par la machine se trouve donc décomposé en tâches qui peuvent avoir entre elles des relations de deux types principaux : séquentialité ou lancement en parallèle sans synchronisation ultérieure.

En effet, les utilisateurs du système CMS se bornent à des tâches séquentielles, comme par exemple le déroulement d'un programme jusqu'à une demande d'entrée-sortie, qui équivaut à la création d'une tâche pour le canal concerné, et attente de la part du programme jusqu'à la fin de l'entrée-sortie, où il y a création (ou réveil) d'une tâche pour l'unité centrale :

tâche unité centrale ⟶ tâche d'entrée-sortie ⟶ tâche unité centrale

Notons que le système découpe lui-même les tâches pour unité centrale pour ses propres besoins, principalement lors de la fin d'un quantum ou lors d'un manque de page virtuelle en mémoire centrale :

tâche unité centrale ⟶ tâche de pagination ⟶ tâche unité centrale

Par contre, dans un système comme OS 360, la création de tâches en parallèle est souvent utilisée :

tâche unité centrale ⟶ tâche d'entrée-sortie ⟶ tâche unité centrale
tâche unité centrale ⟶ .... (traitement d'interruption)

Compte tenu du comportement connu du traitement d'interruption de la machine virtuelle, cette tâche utilisant presque exclusivement la page 0 ne sera pas décrite explicitement. Ceci nous conduit à n'envisager la création de tâches que dans le cas des entrées-sorties, à partir d'une tâche principale pour unité centrale.

Dans la pratique, avant le SIO on arme le mécanisme d'interruption qui lance la suite de la tâche qu'il y ait eu attente ou non.

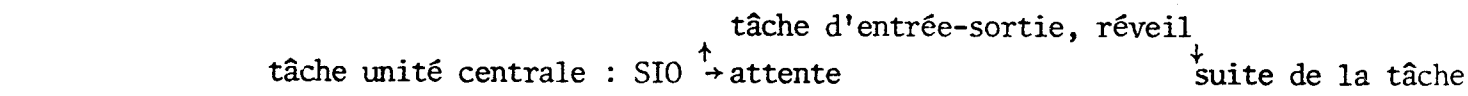
Le modèle le plus élémentaire des instructions non privilégiées consiste à représenter chaque instruction par la liste des pages nécessaires à son exécution dans l'ordre d'extraction des informations par l'unité centrale et par son temps d'exécution.

Nous avons retenu ce modèle en nous autorisant à regrouper plusieurs instructions non privilégiées de telle manière que la liste des pages est l'union de toutes les pages des instructions composantes et le temps d'exécution la somme de tous les temps. Les critères définissant la fin de l'agglomération des instructions jouent un rôle essentiel dans l'adéquation de ce modèle avec le réalité. Outre l'apparition d'une instruction privilégiée, deux autres critères simples peuvent être imaginés :

. dépassement d'un certain nombre (statique ou dynamique) de pages : en dessous de ce nombre, CP doit être insensible à l'ordre exact d'utilisation des pages pour un certain regroupement ;

. dépassement d'une certaine quantité de temps : le contrôle est donné aux machines virtuelles pour un certain temps pendant lequel l'utilisation exacte des pages est indifférente.

De plus, il faut distinguer les instructions privilégiées qui nécessitent une simulation de la part de CP (comme les appels par numéro de fonction, SVC, ou de changements de mode, LPSW) des instructions non privilégiées. Mise à part l'instruction SIO qui crée donc une tâche d'entrée-sortie, un autre cas d'instruction privilégiée doit être détecté séparément : il s'agit de la mise en attente du programme par l'instruction de chargement d'un mot d'état, LPSW, dont le bit d'attente, W, est présent. C'est à l'aide d'une telle instruction que la séquentialité des tâches calcul et entrée-sortie est assurée :



En fait dans la réalité, les machines virtuelles ne peuvent utiliser ce schéma car le temps subissant une distorsion, il se peut que la durée apparente d'une entrée-sortie soit nulle, c'est-à-dire que l'instruction suivant le SIO, qui est souvent le test de son bon déclenchement, intervienne après la fin de l'entrée-sortie : le réveil précèderait la mise en attente ! Un mécanisme de sémaphore permet la remise en ordre de ces actions.

La section 5.1. montrera comment ces hypothèses ont été validées pour divers paramètres à l'aide de programmes étalons.

Finalement, le modèle des tâches de machines virtuelles se compose d'une séquence de "macro-instructions" des types suivants :

· non_privilégié : type (=1), liste de pages, nombre d'instructions

· entrée-sortie_effective : type (=2), liste de pages, nombre de pages bloquées, numéro de périphérique, liste de transferts

· mise_en_attente : (LPSW avec bit W) : type (=3), liste de pages

· autres_instructions_privilégiées : type (=4), liste de pages, code instructions.

La liste de pages comprenant :

nombre de pages, ..., indication de changement, numéro de page, .. (écriture)

et celle de transferts :

nombre de transferts, ..., cylindre, adresse début, longueur, ...

ces différentes macro-instructions définissent l'interface entre le modèle de CP demandant la macro-instruction suivante à exécuter pour la machine virtuelle numéro i et le modèle de tâche qui peut fabriquer cette macro-instruction de diverses manières :

1) traces de programmes réels (cf. validation § 5.1),

2) programmes composés ayant certaines caractéristiques (cf. expériences § 5.3.),

3) programmes engendrés d'une manière probabiliste.

La méthode 3) n'a pas été utilisée car le nombre de paramètres en jeu rendait déjà difficile leur étude déterministe.

4.2.3. Relations entre sélection et mise à jour

La procédure de sélection d'une tâche par une ressource active est complétée par les procédures de mise à jour des autres ressources qui peuvent créer des tâches pour la première. Par exemple, une politique de choix FIFO peut être réalisée par deux procédures qui co-opèrent de la même façon que les "producteurs" et "consommateurs" (cf. Dijkstra).*

```
procédure sélection (m) ;
   début P(s) ;
      m := T(i) ;
      i := i + 1 ;

   fin ;
```

```
procédure mise à jour (m) ;
   début
      T(j) := m ;
      j := j + 1 ;
      v(s ) ;
   fin ;
```

où i et j sont initialisés à i_o et s à zéro et la portion (i,j) du tableau T représente la file d'attente des tâches.

De même, si la politique de choix est basée sur la priorité des tâches, les procédures peuvent s'écrire :

```
procédure sélection (m)
   début P(s) ;
      P(mutex) ;
         m := T(i) ;
         i := i + 1 ;
      v(mutex) ;

   fin ;
```

```
procédure mise à jour (m) ;
   début P(mutex) ;
      T(j) := m ;
      j := j + 1 ;
      ordonner (i,j) ;
      v(mutex) ;
      v(s) ;
   fin ;
```

---

* Voir au § 3.1.1. la définition des procédures p(s) et v(s).

ou encore :

```
procédure sélection (m) ;
   début P(s) ;
      P(mutex) ;
      ordonner (i,j) ;
      m := T(i) ;
      i := i + 1 ;
      v(mutex)
   fin ;
```

```
procédure mise à jour (m) ;
   début P(mutex) ;
      T(j) := m ;
      j := j + 1 ;
      v(mutex) ;
      v(s) ;

   fin ;
```

suivant que l'ordonnancement se fait dans la procédure de mise à jour ou dans celle de sélection (par ordonner (i,j)).

Pour que l'effet soit le même dans les deux cas, il faut et il suffit que les priorités restent ordonnées de la même manière dans le temps.

Si après une attente t l'on appelle $p(a_i,t)$ la priorité de la $i^{ème}$ tâche, entrée en file d'attente avec $p(a_i,0) = a_i$, on doit avoir :

$$[\forall t_2 > t_1,\ p(a_i,t_1) \leqslant p(a_j,t_1) \Rightarrow p(a_i,t_2) \leqslant p(a_j,t_2)]$$

D'un autre côté, on aimerait que les valeurs de la propriété ne dépendent pas du "passé", c'est-à-dire que :

$$[\forall t_1, \forall t_2 > 0,\ p(a_i,t_1+t_2) = p(p(a_i,t_1),t_2)]$$

En particulier, les fonctions de la forme :

$$p(a_i,t) = a_i e^{kt}$$

$$p(a_i,t) = a_i + k't \quad (k' \geqslant 0)$$

vérifient ces deux propriétés et donnent plus de souplesse pour décider de l'instant d'ordonnancement. La seconde forme a été analysée par Kleinrock [51] lorsque a_i est proprotionnel au service reçu. Le choix du moment d'ordonnancement peut dépendre des facteurs suivants :

. une des deux ressources effectuant ces procédures est saturée, l'autre peut donc faire ce travail. Par exemple, une unité centrale très saturée ne fera pas l'ordonnancement des tâches (cas des systèmes sur CDC 6600). Au contraire, lorsqu'un canal de pagination est saturé, l'unité centrale se charge du choix des pages "à vider".

. Des tâches peuvent quitter la file d'attente avant d'être servies. Dans ce cas, le nombre d'entrées en file d'attente peut dépasser notablement celui des services : on a intérêt à ordonnancer juste avant ce service, c'est-à-dire lors d'une procédure sélection. Par exemple, les événements prévus en file "d'échéance" lors d'une simulation temporelle sans structure de processus, peuvent être décommandés (ou retardés) avant leur exécution. De même, dans un système CP, les utilisateurs peuvent quitter la file d'attente de l'unité centrale pour demander un service "CP" (renseignements, messages), ce qui reste un cas peu fréquent.

Lorsque l'ordonnancement dépend de l'instant où il est effectué, les différentes solutions mettent en relief un phénomène analogue à celui de la préemption qui consiste à interrompre une tâche en cours dès l'apparition d'une tâche plus prioritaire.

Dans la pratique, la préemption est utilisée pour les tâches extrêmement prioritaires qui sont, soit certaines tâches des systèmes "temps réel", soit le traitement des interruptions de fin d'entrée-sortie. Dans les grands systèmes à multiprogrammation ou à temps partagé, la préemption n'est pas forcément utilisée pour les tâches de l'unité centrale car "l'inertie" de la commutation peut être importante à cause notamment des registres de travail, des tables de correspondance entre adresses virtuelles et physiques (cf. CII 10070), des anté-mémoires utilisées (cf. IBM 370-158, 370-168). On attend plutôt la fin d'un quantum de temps ou un événement logique pour entreprendre la tâche la plus prioritaire, ou plus simplement le nom des commandes invoquées qui déterminera par analyse topologique les ressources en question (cf. Leroudier [55], Borrione [16]).

4.2.4. Types de ressources envisagées et leurs relations

Cinq types de ressources ont été envisagés dans ce modèle :

1) l'unité centrale (U.C.)
2) le canal d'entrée-sortie
3) le canal de pagination
4) le canal multiplexeur (pour consoles)
5) la ressource factice.

Le canal de pagination est traité de façon séparée des autres canaux d'entrée-sortie pour deux raisons principales :

. le système CP utilise, en mode normal, un canal spécifique pour le tambour de pagination ;

. la procédure de mise à jour pour cette ressource est spécifique (algorithme de remplacement), ainsi que celle de sélection ("file d'attente par secteur", cf. Coffman [26]).

Néanmoins, en cas d'anomalie, la pagination peut se faire sur disques, la sélection étant FIFO sur l'ensemble des tâches du canal. La distinction que nous avons établie devient donc parfaitement inutile.

D'une façon générale, certains paramètres "quantitatifs" sous-entendent une "organisation" du cas général : par exemple, le nombre d'unités centrales peut supposer une organisation "banalisée" où les tâches sont mises en commun, alors qu'un second tambour de pagination peut impliquer certaines restrictions sur l'algorithme de remplacement.

Nous appelerons ressources équivalentes des ressources capables d'exécuter les mêmes tâches et qui les sélectionnent dans le même ensemble, ce qui impliquera, par exemple pour ces canaux, que les périphériques soient accessibles par chaque canal.

Les opérations que font de telles ressources sur l'ensemble des tâches doivent être faites en exclusivité. Le cycle des processus précède la procédure de sélection par un "p" sur sémaphore de mutuelle exclusion, attaché à l'ensemble des tâches. Les mises à jour sur cet ensemble font de même, mais peuvent réveiller certaines ressources en attente. Pour la correction de cette coopération, il faut naturellement que le prédicat de réveil soit exact afin que les ressources en sélection traversent entièrement les "sections critiques".

Dans les systèmes actuels, la décision de sélection peut être partagée :

. lors de la mise à jour, une éventualité de réveil est signalée,

. la sélection examine si cette dernière peut se tranformer en réalité.

Une telle situation peut être traitée à l'aide de plusieurs sémaphores, ou, lorsque la situation est celle de processus imbriqués (ou "hiérarchisés", cf. Anceau [2]) par des opérations p et v qui s'induisent à différents niveaux.

Les consoles d'entrée-sortie sont des ressources n'admettant qu'une tâche dont la durée dépend des messages imprimés ainsi que du délai de réflection.

Compte tenu du faible taux d'activité normale des autres périphériques attachés au canal multiplexeur, lecteur de cartes, imprimante, bandes, ceux-ci ne sont pas représentés et le canal multiplexeur est donc modelé comme un ensemble de ressources indépendantes (les consoles) ayant mêmes caractéristiques.

La ressource factice est artificiellement introduite pour grouper les machines virtuelles en attente. Le réveil de tâches est remplacé par une transition de la ressource factice à une ressource réelle.

Finalement, on aboutit à la structure de la figure 4.2.

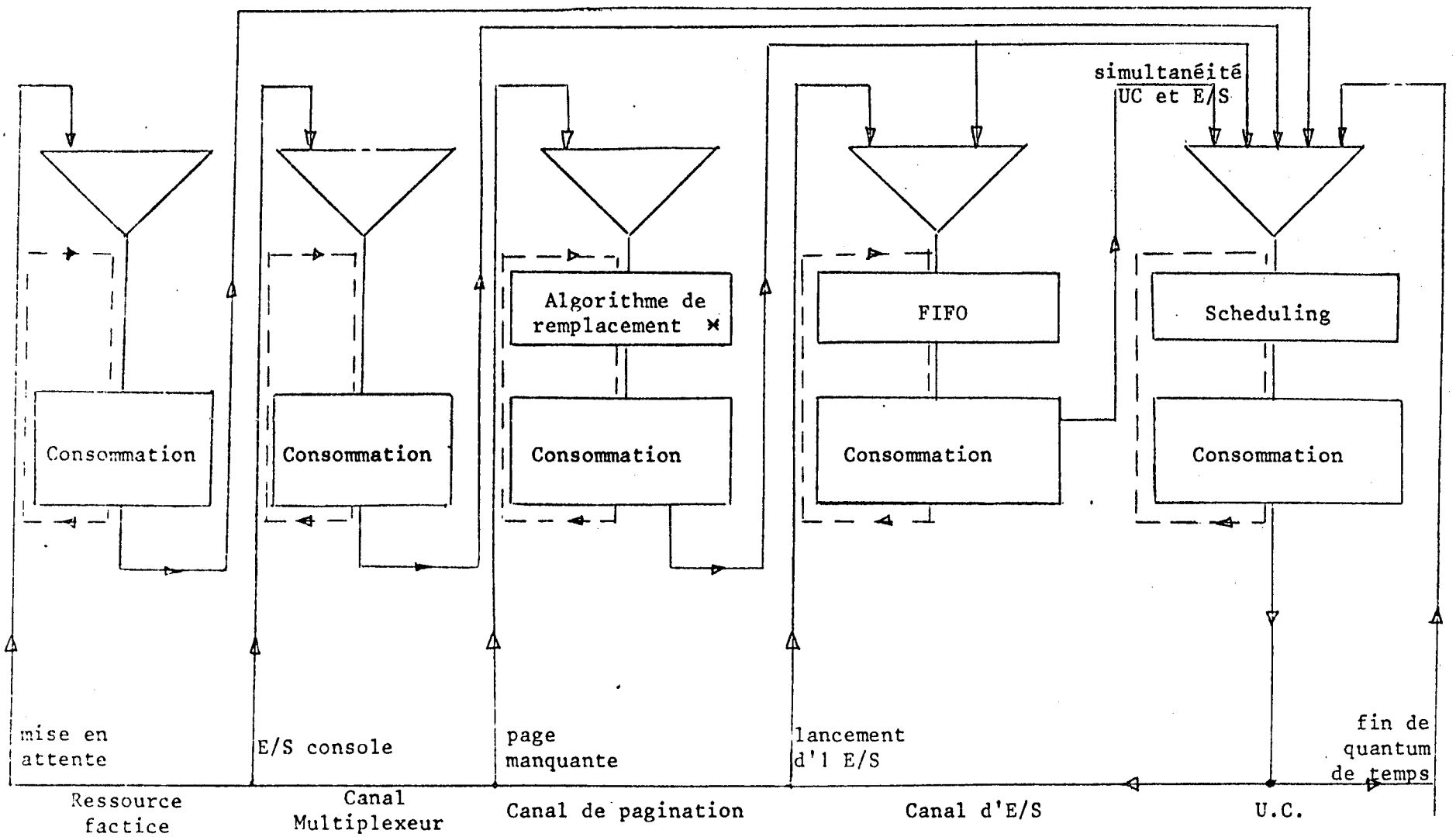


FIGURE 4.2. - Modèle de CP-67 par processus attachés aux ressources physiques

* FIFO ou attente par secteur

Examinons plus en détail le cycle des ressources des quatre premiers types.

4.3. Cycle de l'unité centrale

4.3.1. Procédure d'exécution

Les "macro-instructions" (décrites au § 4.2.1.b.) sont fournies une à une à la procédure d'exécution pour simuler le déroulement du programme d'une machine virtuelle, par une procédure de modèle des programmes. Cette simulation comprend la recherche des pages référencées, l'indication éventuelle de modification de ces dernières et la diminution correspondante d'un compteur de temps, le "timer", servant à la mise à jour de l'horloge. La cause de la fin de cette consommation peut être :

1) interruption à la valeur 0 du compteur de temps,
2) ordre de lancement d'une entrée-sortie (SIO),
3) une page référencée, au moins, n'est pas en mémoire centrale,
4) mise en attente de l'unité centrale virtuelle.

Remarques :

1. Une interruption du timer peut se produire au milieu d'une macro-instruction. Ce fait, ainsi que le temps restant, seront retenus pour reprendre la partie non exécutée de la macro-instruction.

2. Le fait de considérer plusieurs pages manquantes à la fois alors qu'en réalité elles se manifestent une par une, risquait de déformer le modèle. Dès que l'espace alloué est environ le double du nombre de pages maximum permis dans une macro-instruction (que nous appelerons facteur de groupage), cette distorsion disparaît quant au nombre de pages manquantes. L'activité des ressources n'est perturbée que localement dans le temps, puisqu'en fait tous les manques de pages se produisent, mais l'allure de la file d'attente sur le canal de pagination est fortement changée.

En conséquence, cette dernière variable n'a pas été prise en considération. Afin d'assurer une bonne prédiction lorsque l'espace d'une tâche est du même ordre (ou moindre) que le facteur de groupage, un raffinement du modèle de la macro-instruction a été utilisé. Il sera décrit au § 5.1.1. avec l'expérience de validation qui montre son utilité.

3. Le traitement d'interruption d'une entrée-sortie, qui n'apparaît que par le temps d'unité centrale consommé, est toujours reporté en fin de procédure d'exécution. Ce décalage ne provoque qu'une déformation locale et s'est révélé négligeable dans les expériences de calibration (cf. § 4.3.2.)

4. L'unité de temps choisie est 100 µs pour le modèle général. Néanmoins, l'unité centrale décompte le nombre d'instructions de chaque tâche les exécute et les convertit en temps à l'aide d'un temps moyen d'exécution d'une instruction. Ce paramètre doit intégrer des phénomènes comme le conflit d'accès à la mémoire entre unité centrale et canaux, ou comme les cycles des différents modules de mémoire centrale.

4.3.2. Procédure de sélection

Cette procédure, ainsi que les procédures de mise à jour, créant ou réveillant des tâches pour unité centrale, opère sur deux partitionnements de l'ensemble des tâches :

P1 : partition en "type de travail" :

On distingue trois classes ou niveaux correspondant aux utilisateurs :

1) interactif (en fait, ayant fait une interaction récemment),
2) faisant plutôt beaucoup d'entrées-sorties,
3) faisant plutôt beaucoup de calculs.

En pratique, lors d'une interaction (entrée ou sortie à la console), l'utilisateur est classé comme interactif, puis après l'usage d'un certain temps d'unité centrale (par exemple, pour la version 3.0 de 400 ms cumulées ou 50 ms d'une traite), il est déclassé au niveau 2, puis après l'usage d'un second laps de temps d'unité centrale (2 s cumulées ou 50 ms d'une traite, toujours pour la version 3.0), il est déclassé au niveau 3.

Il convient de noter que les entrées-sorties, autres que pour la console ainsi d'ailleurs que la pagination, n'affectent qu'indirectement leur classement en empêchant la consommation du "petit" quantum de 50 ms en une seule fois.

Ces niveaux correspondent à des priorités de traitement décroissantes.

P2 : partition pour étaler la charge :

A chaque niveau sont distingués les utilisateurs "actifs" (seuls pris en considération pour la sélection effective) et les "inactifs" (les autres). Un nombre maximum d'utilisateurs actifs est respecté pour le niveau 1 et un autre pour l'ensemble des niveaux 2 et 3. Ces nombres sont basés sur l'hypothèse d'un usage moyen de la mémoire dans chacun de ces niveaux et dépend de la taille de la mémoire réelle de CP-67 (voir tableau).

Taille de la mémoire réelle	256 K	512 K	1024 K	2048 K
Nombre d'utilisateurs du niveau 1	3	6	9	12
Nombre d'utilisateurs des niveaux 2 et 3	1	3	6	20

Les utilisateurs inactifs attendent une place libre dans le sous-ensemble actif de leur niveau pour y entrer, dans l'ordre de leur dernière entrée dans ce niveau.

Ces deux partitions donnent la structure suivante :

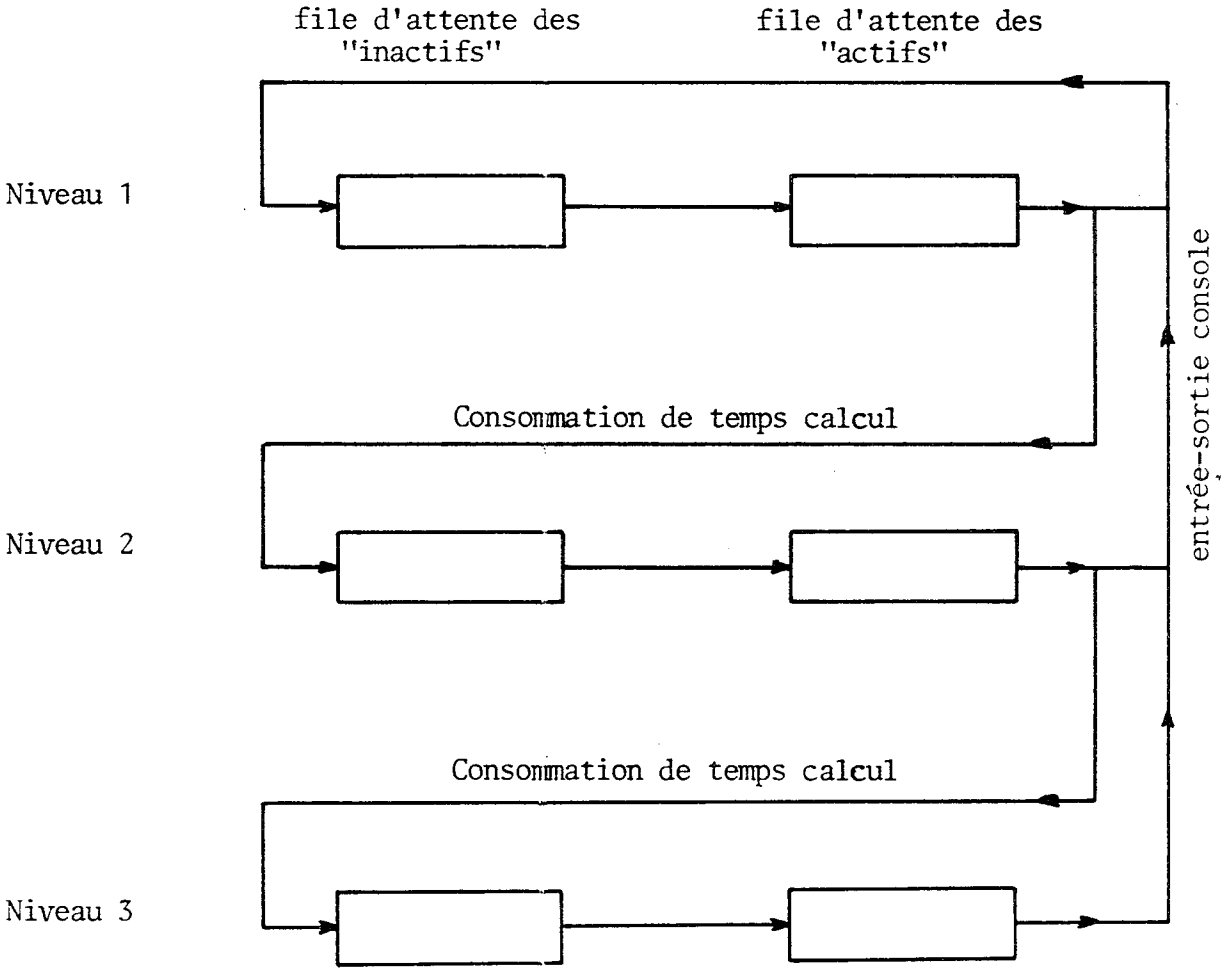


Figure 4.3.

Rappelons les effets voulus par cet algorithme :

les machines virtuelles actives se partagent l'usage des ressources dans le futur immédiat. Elles prendront, de préférence, les pages des machines inactives (cf. l'algorithme de pagination) : leur nombre de pages aura tendance à croître fortement en comprenant des pages désormais inutiles qui seront expurgées lorsque ces machines seront inactives. Rien, néanmoins, ne garantit que le passage à la compétition mutuelle entre machines actives ne se fera pas.

4.3.3. Procédure de mise à jour

La cause de la fin de la procédure de l'exécution détermine l'action à entreprendre :

1) sur épuisement du compteur de temps, la tâche est ré-introduite comme candidate à l'unité centrale avec modification de son état conforme à l'algorithme de sélection (cf. § 4.3.1.) ;

2) sur lancement d'une entrée-sortie sur canal simple, une tâche pour le canal réel correspondant est créée. A cette occasion, il faut effectuer une traduction des programmes canaux. En effet, les entrées-sorties se déroulent à l'aide d'adresses réelles. Il faut donc, d'une part, s'assurer que les pages virtuelles sont présentes en mémoire centrale, d'autre part, "éclater" en plusieurs morceaux un transfert qui ne porte plus sur des zones contigües de mémoire réelle. Comme les disques sont partagés, chaque transfert est précédé d'un ordre de positionnement du bras. Ce positionnement est séparé du transfert pour des raisons d'efficacité (cf. § 4.4.1.). Si certaines pages ne sont pas présentes en mémoire centrale, la tâche d'entrée-sortie et celle pour unité centrale qui attend un code de réception sont mises en attente et une tâche de pagination est créée et mise dans l'ensemble des candidates au canal de pagination. Les pages nécessaires à l'entrée-sortie sont "bloquées" en mémoire centrale, c'est-à-dire qu'elles ne sont plus susceptibles d'être choisies par l'algorithme de remplacement. Une liste de ces pages est conservée par la tâche d'entrée-sortie pour le débloquer.

3) Sur page manquante, une tâche de pagination est créée et mise dans le même ensemble de candidats. Il convient de noter que la création d'une tâche de pagination comprend la détermination des adresses physiques de transferts, en particulier celles de la mémoire centrale, par l'algorithme de remplacement. Celui-ci logiquement aurait pu être incorporé à l'algorithme de sélection du canal de pagination. Comme le temps pris est assez long (2 ms), il est nécessaire d'anticiper cette décision. Cela peut aussi bien se faire en synchronisme avec le cycle du canal de pagination, mais la solution adoptée par CP reste la plus simple. Si un processeur indépendant se chargeait de cette tâche, le principal effet sur le modèle serait de rendre nul le temps d'exécution de l'algorithme de remplacement qu'il soit incorporé à l'unité centrale ou au canal de pagination.

4) Sur mise en attente de la tâche, effectuée par le chargement du registre d'état avec indication d'attente (bit W), la tâche est mise dans l'ensemble des tâches en attente, conceptuellement identifiée à l'ensemble des candidats d'une ressource "factice". La fin d'une entrée-sortie réintroduira cette tâche comme candidate pour l'unité centrale. Il est intéressant de noter que la programmation ne peut se faire de manière aussi simpliste que : lancement d'une entrée-sortie, mise en attente car l'entrée-sortie peut à la limite avoir une durée virtuelle nulle. La mise en attente survenant après la fin de l'entrée-sortie risque alors d'être éternelle ! Cette durée nulle peut provenir soit du fait que l'entrée-sortie est purement virtuelle, (simulée par le "spooling" et ne consistant qu'en un transfert vers un tampon de la mémoire centrale par exemple), soit du fait d'une commutation de tâches intervenant à ce moment (lorsqu'il y a d'autres demandes sur le canal réel concerné). En fait, il faut préparer l'aboutissement de la séquence d'interruption de fin d'entrée-sortie avant le lancement de cette dernière.

4.4. Cycle du canal "simple" d'entrée-sortie

4.4.1. Procédure de sélection

La règle de sélection adoptée par CP est celle FIFO ("First In First Out") et est réalisée par une simple liste de laquelle la procédure de sélection prélève la tête. Les procédures de mise à jour qui introduisent de nouveaux candidats le font par la fin de cette liste, en réveillant le canal si cette liste était vide.

La simplicité de cette règle se justifie compte tenu du faible niveau de multiprogrammation (inférieur à 10) et du grand nombre de disques utilisés (de l'ordre de 16) qui limite la longueur de la file d'attente pour chaque disque (cf. Lynch [57]) et rend inutile l'application de stratégies plus élaborées (cf. Teorey et Pinkerton [80]). Néanmoins, la simultanéité possible des mouvements de bras de disques et des transferts de données est exploitée en initialisant tous les mouvements de bras demandés sur des disques différents de ceux des demandes antérieures en attente entre deux transferts. Ceci se fait à l'occasion de la "traduction" des programmes canaux lors de la procédure de mise à jour de l'unité centrale (cf. § 4.3.3.).

4.4.2. Procédure d'exécution

Elle simule l'occupation d'un canal simple d'entrée-sortie. Nous nous sommes bornés à des périphériques de type disques (de type IBM 2314), mais d'autres périphériques seraient facilement simulables.

Cette procédure détermine le temps que prendra l'entrée-sortie, somme du temps de mouvement du bras (calculé), du temps d'accès rotationnel (approché par sa valeur moyenne dans notre modèle) et du transfert lui-même (calculé).

Des raffinements possibles à la version actuelle pourraient inclure l'usage de variable aléatoire pour le temps d'accès rotationnel. Ceci n'a pas semblé utile dans un premier temps.

Enfin, cette procédure rend la machine virtuelle candidate pour l'unité centrale immédiatement puisque cette dernière peut opérer en simultanéité avec les canaux d'entrée-sortie. Cette possibilité, non nécessaire logiquement, est utilisée pour des systèmes comme OS 360 utilisés seuls sous CP.

4.4.3. Procédure de mise à jour

Elle diminue d'un le compte de blocage des pages de l'entrée-sortie (car une page peut être bloquée pour plusieurs entrées-sorties). Cette procédure reflète ensuite l'interruption de fin d'entrée-sortie et un traitement standard de la manière suivante :

. si les pages 0, 1, 2, 3, de la machine virtuelle sont en mémoire, un temps forfaitaire est ajouté à l'activité de l'unité centrale ;

. si une ou plusieurs de ces pages manquent, une tâche de pagination est créée et mise dans l'ensemble des candidates au canal de pagination. La réflection se fera à la fin de cette tâche. Cette réflection provoque le réveil éventuel de la tâche unité centrale de la machine virtuelle. Dans le cas où il y a plusieurs entrées-sorties simultanées, la fin de la première réveille la tâche unité centrale. Ce cas n'intervient pas dans la simulation d'utilisateurs de CMS.

4.5. Cycle du canal de pagination

4.5.1. Procédure de sélection

L'algorithme de remplacement invoqué par les créations de tâches pour le canal de pagination choisit la page à remplacer d'abord parmi les utilisateurs "inactifs" (cf. § 4.3.2.P2.), puis parmi les "actifs". A l'intérieur de ces catégories un effet FIFO est recherché (CP version 3.0). La version suivante (3.1) de CP utilise les indicateurs de page non utilisée, remis périodiquement à "non utilisé", pour sélectionner les pages à enlever. Un effet LRU (Least Recently Used) par période est ainsi réalisé. Au § 5.3. des expériences de comparaison de ces deux algorithmes sont commentées.

Il est intéressant de noter que dans CP chaque machine virtuelle n'a au plus qu'une demande de page alors que nous en considérons une liste. Cette simplification demandait à être justifiée par les expériences (cf. § 5.1.1.).

Les pages en cours de transfert sont dites "en transit" et l'algorithme de remplacement recherche tout d'abord si la ou les pages demandées ne sont pas "en transit", ce qui nécessite une synchronisation supplémentaire et s'interdit bien sûr de remplacer ces pages bloquées ou en transit. Cette sélection de la page à remplacer est effectuée en même temps que la création de la tâche.

Dans la version 3.0, la règle de sélection était FIFO. Afin d'éliminer le temps d'accès rotationnel, lorsque la charge est suffisante, les versions suivantes ont utilisé la règle de sélection "par secteur" de tambour (cf. Coffman [26]).

Une piste du tambour est divisée en secteurs angulaires pouvant chacun contenir une page. Une file d'attente des demandes est maintenue par secteur. Le service consiste à chaîner les premières demandes des files d'attente des secteurs dans l'ordre de défilement devant les têtes de lecture/écriture. Sous une charge importante, on élimine ainsi le délai rotationnel et le débit effectif du tambour approche le débit de transfert.

Notons qu'entre transferts successifs, les pistes concernées changent ainsi que les adresses en mémoire centrale.

Pour CP, la sectorisation s'est faite sur deux pistes de tambour qui peuvent accueillir cinq pages, le principe restant le même.

Lorsque plusieurs tambours sont connectés sur le même canal, on perd une partie de l'avantage de la sectorisation lors des commutations d'un tambour à l'autre, dont il faut définir à quelles conditions elles se feront.

4.5.2. Procédure d'exécution

Les pages amenées en mémoire centrale sont marquées "en transit" de façon à bloquer la page physique jusqu'à la fin du transfert, sans que l'on puisse encore l'utiliser (cas des pages partagées de CMS), ni que l'on demande deux fois le transfert de ces pages. Suivant l'algorithme de service, le temps est décompté avec ou sans délai rotationnel.

4.5.3. Procédure de mise à jour

Elle transforme les pages "en transit" en pages présentes, disponibles à la fois pour l'unité centrale et pour ... l'algorithme de remplacement. Si d'autres tâches attendaient ces pages pour continuer, elles sont réveillées. Suivant que la tâche de pagination était issue ou non d'une demande d'entrée-sortie, on réveille soit la tâche d'entrée-sortie, soit la tâche pour unité centrale. Ces réveils peuvent éventuellement activer les ressources correspondantes si elles étaient inactives.

4.6. Cycle du canal multiplexeur

Il est simulé comme autant de canaux simples que de consoles. La sélection est immédiate car il n'y a qu'un candidat au plus et la consommation se borne au décompte du temps passé en sortie ou en entrée. En sortie, la vitesse est celle du terminal (10 à 30 caractères par seconde), alors qu'en entrée, la temps de réflection est donné par le modèle de programme (habituellement de 5 à 60 secondes). La mise à jour réveille les tâches unité centrale habituellement en attente pour une telle entrée-sortie.

4.7. Données numériques sur le système CP/CMS

Dans l'environnement du Centre Scientifique IBM de Cambridge (USA), le système CP-67 a été soumis à des mesures depuis 1970. Y. Bard [7] a étudié un modèle de régression linéaire reliant le temps que CP-67 utilise en mode superviseur en fonction de différents événements observables. Par exemple, une des équations envisagées est :

$$CP=CO/C1\times T+C2\times NVSIO+C3\times NVMIO+C4\times NSPOOL+C5\times NPAG$$

où

CP est le temps en mode superviseur,

CO, C1, C2, C3, C4, C5, sont les coefficients à évaluer selon les moindres carrés,

T est le temps en mode problème délivré aux utilisateurs,

NVSIO est le nombre d'entrées-sorties sur les canaux sélecteurs (tambours et disques),

NVMIO est le nombre de celles sur canal multiplexeur (console, imprimante, lecteur et perforateur de cartes),

NSPOOL est le nombre de celles simulées en deux temps asynchrones (imprimante, lecteur et perforateur de cartes),

NPAG est le nombre de pages apportées en mémoire.

Les valeurs sont mesurées, d'une part sur le temps total en mode superviseur et, d'autre part, sur celui non dû à l'événement lui-même (et donc non facturé) mais à la machinerie de CP-67 ("overhead").

C2 = 7.9 MS	OC2 = 0.6 MS
C3 = 2.6 MS	OC3 = 0.3 MS
C4 = 4.7 MS	OC4 = 1.1 MS
C5 = 2.0 MS	OC5 = 0.7 MS

Les coefficients C0 et C1 ne montrent pas la même constance selon les périodes envisagées, certainement parce que des événements non pris en compte doivent avoir une influence sur eux (par exemple, nombre d'instructions privilégiées simulées).

On peut alors déduire le coût de certains algorithmes :

pagination	= C5-OO5	= 1.3 MS
aiguillage	= 1/2(OC2+OC5)	= 0.65 MS
traduction des programmes canaux	= C2-OC2	= 7.3 MS

Ces valeurs ont été utilisées dans la simulation pour estimer le coût des algorithmes actuels, prévoir leur variation avec certains paramètres, évaluer le coût de leurs variantes.

Il convient d'assortir quelques remarques à ces mesures :

1) En proportion, la traduction des programmes canaux utilise plus 50 % du temps superviseur. Compte tenu du fait que CMS n'utilise pratiquement que des fichiers sur disques organisés en enregistrements de longueur fixe (800 octets), les versions suivantes de CP ont utilisé un mécanisme de "traduction accélérée" déclenché par un "diagnose" (instruction privilégiée de maintenance) plutôt que par un SIO normal. Cette modification aux systèmes CP et CMS a pratiquement divisé le coût de ce service par 10 ! C'est certainement une des améliorations les plus spectaculaires de CP, mais qui enfreint le principe des machines virtuelles conformes à des machines réelles. En fait, la machine virtuelle tend à devenir une entité logique et CP rejoint ainsi les concepts de MTS ou MULTICS !.

2) L'algorithme de remplacement mesuré par un instrument matériel, le BCU ("Basic Counter Unit") ne consomme que 0.5 ms des 1.3 ms associées à l'activité de pagination. Le reste concerne surtout la création de la tâche de pagination et le réveil de la tâche unité centrale ou canal d'entrée-sortie en fin de la première.

CHAPITRE 5

LES EXPERIENCES DE SIMULATION DE CP/CMS

Conformément à la méthodologie que nous avons évoquée, le modèle doit être validé avant de servir d'outil de prédiction.

5.1. Validation du modèle

Parmi toutes les simplifications apportées au modèle, certaines semblent indispensables compte tenu des informations d'entrée (ignorance des conflits d'accès à la mémoire, temps moyen de durée d'une instruction), d'autres ont été volontairement introduites car les phénomènes écartés semblaient négligeables (regroupement des instructions, élimination des périphériques gérés par symbionts). Des expériences de complexité croissante ont été entreprises afin de valider les divers aspects du modèle.

5.1.1. Validation de la pagination sur un seul utilisateur

L'agglomération d'instructions qui a simplifié le comportement du programme quant à l'adressage aurait pu avoir une influence sur la pagination du système CP/CMS. Afin de mettre en évidence, une expérience de simulation d'un seul utilisateur sous CP a été entreprise.

Deux exécutions de programmes particuliers ont été analysées à l'aide d'un simulateur logiciel, afin d'en extraire la trace de références à la mémoire. Ces traces permettent de définir le nombre exact de défauts de page que ces programmes font dans un espace fixe de n pages, soumis à la politique de remplacement de CP (Hatfield [40]). Par ailleurs, à partir de ces traces, on peut agglomérer les instructions jusqu'à une limite de x pages et injecter les données ainsi réduites au simulateur.

La figure 5.2.a. compare les deux résultats pour une compilation du langage AED de 500 000 instructions référençant 47 pages et pour un facteur de groupage, x, de 5. L'exactitude de la simulation est parfaite.

Les figures 5.2.b. et 5.2.c. comparent les résultats des deux expériences pour un facteur de groupage de 10, l'une pour la compilation précédente, l'autre pour une exécution de l'assembleur OS XF de 1 800 000 instructions sur 60 pages.

De légères différences apparaissent lorsque la taille de la mémoire disponible approche 10. Afin de surmonter ce type de difficulté, Greenberg et Tillman [14] ont imaginé de raffiner le modèle où les références sont groupées lorsque la mémoire disponible tend vers le facteur de groupage : seule une fenêtre de trois ou quatre pages est utilisée à un instant donné, la fenêtre glissant régulièrement dans le temps le long de l'ensemble des pages de la macro-instruction :

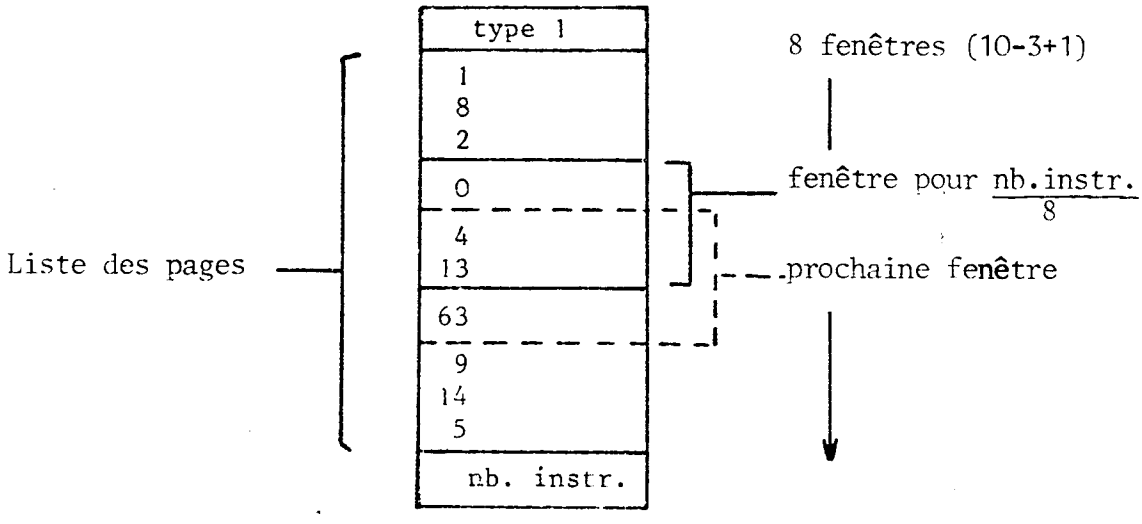


Figure 5.1.a.

Ceci revient à supposer que l'occupation de l'espace dans le temps est décrite plus finement :

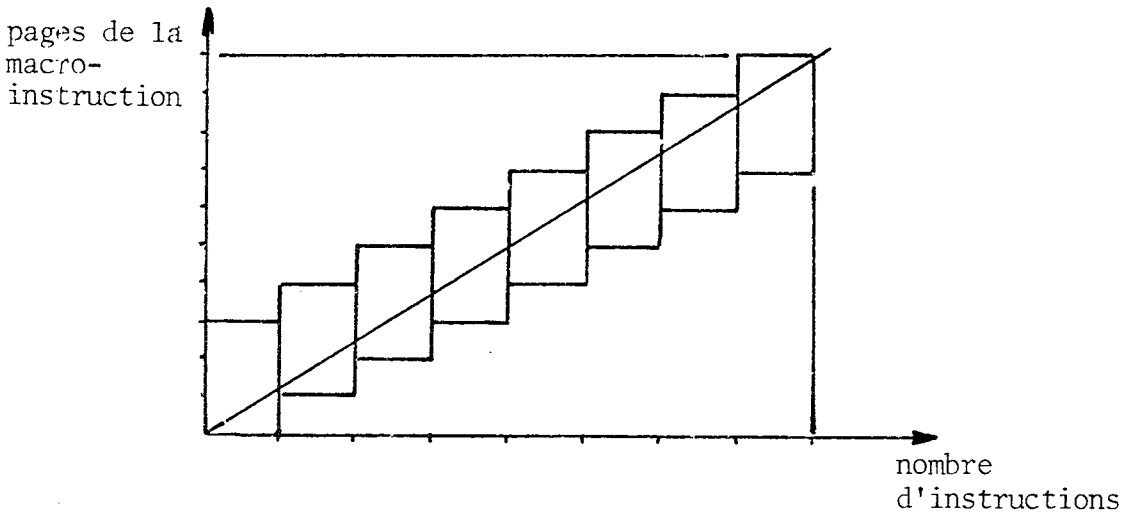
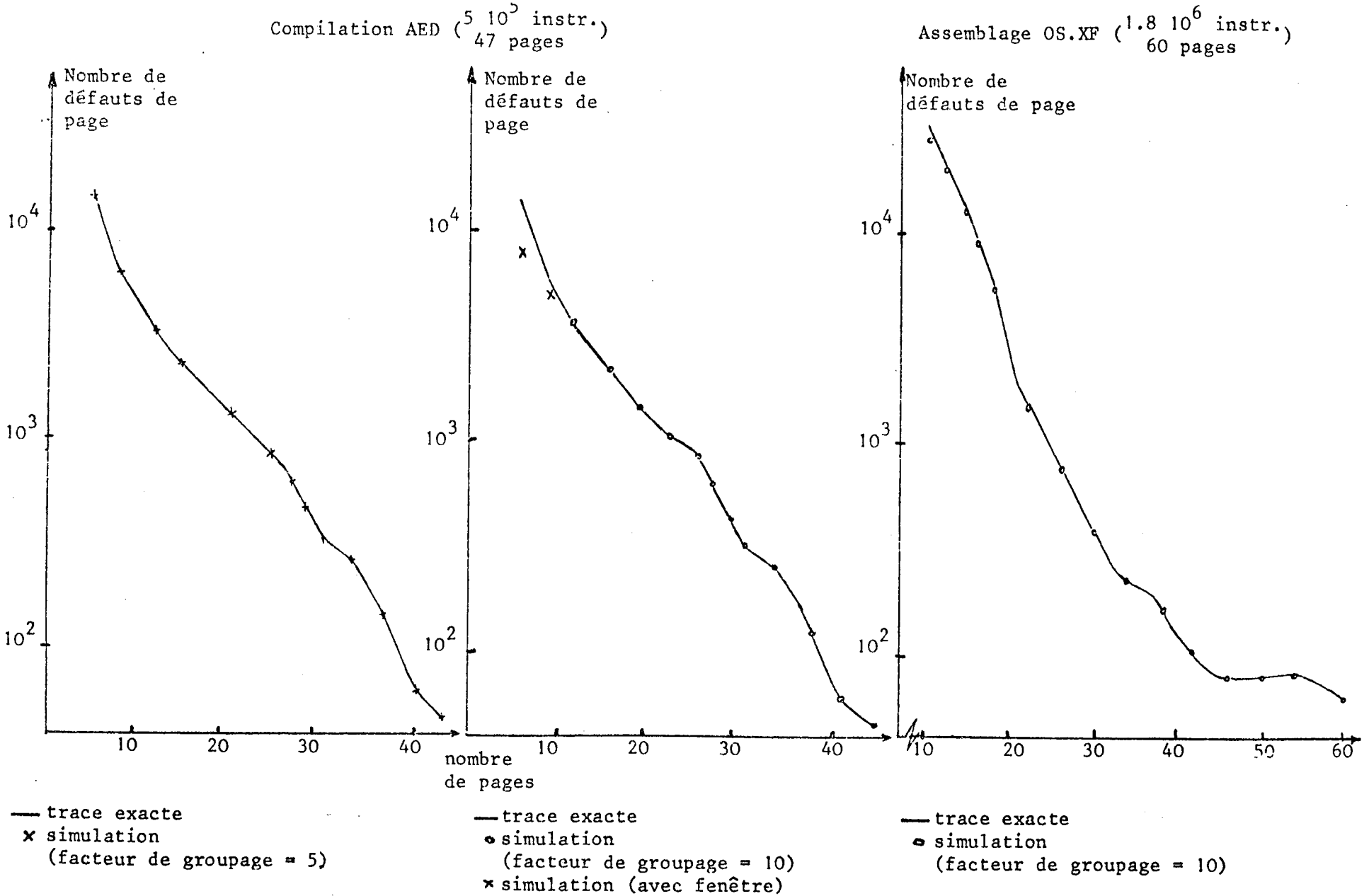


Figure 5.1.b.

A la place du rectangle global d'espace-temps, l'occupation est réduite aux rectangles entourant la diagonale.

Ce modèle appliqué aux données précédentes a fourni les points (x) de la figure 5.2.b. Quoique différentes du nombre réel de défauts de page, les approximations obtenues semblent satisfaisantes.



Figures 5.2.a., 5.2.b., 5.2.c. - Validation mono-utilisateur (facteur de groupage)

5.1.2. Validation de la pagination avec plusieurs utilisateurs

Les expériences précédentes ont confirmé que le modèle d'adressage du programme était satisfaisant. Néanmoins, l'algorithme de pagination de CP peut se trouver légèrement perturbé par le groupage des pages et induire des différences dans le séquencement des machines virtuelles.

Des expériences où plusieurs utilisateurs étaient présents ont été entreprises afin d'étudier ce phénomène.

L'instrument de prélèvement de mesures sur le système réel (version 3.1) se compose d'une machine virtuelle qui active un programme, "DUSETIMR" à des intervalles variant de 5 à 25 s, avec une imprécision dans le déclenchement provenant du fait qu'elle est ordonnancée comme une autre par CP et que sa priorité est faible.

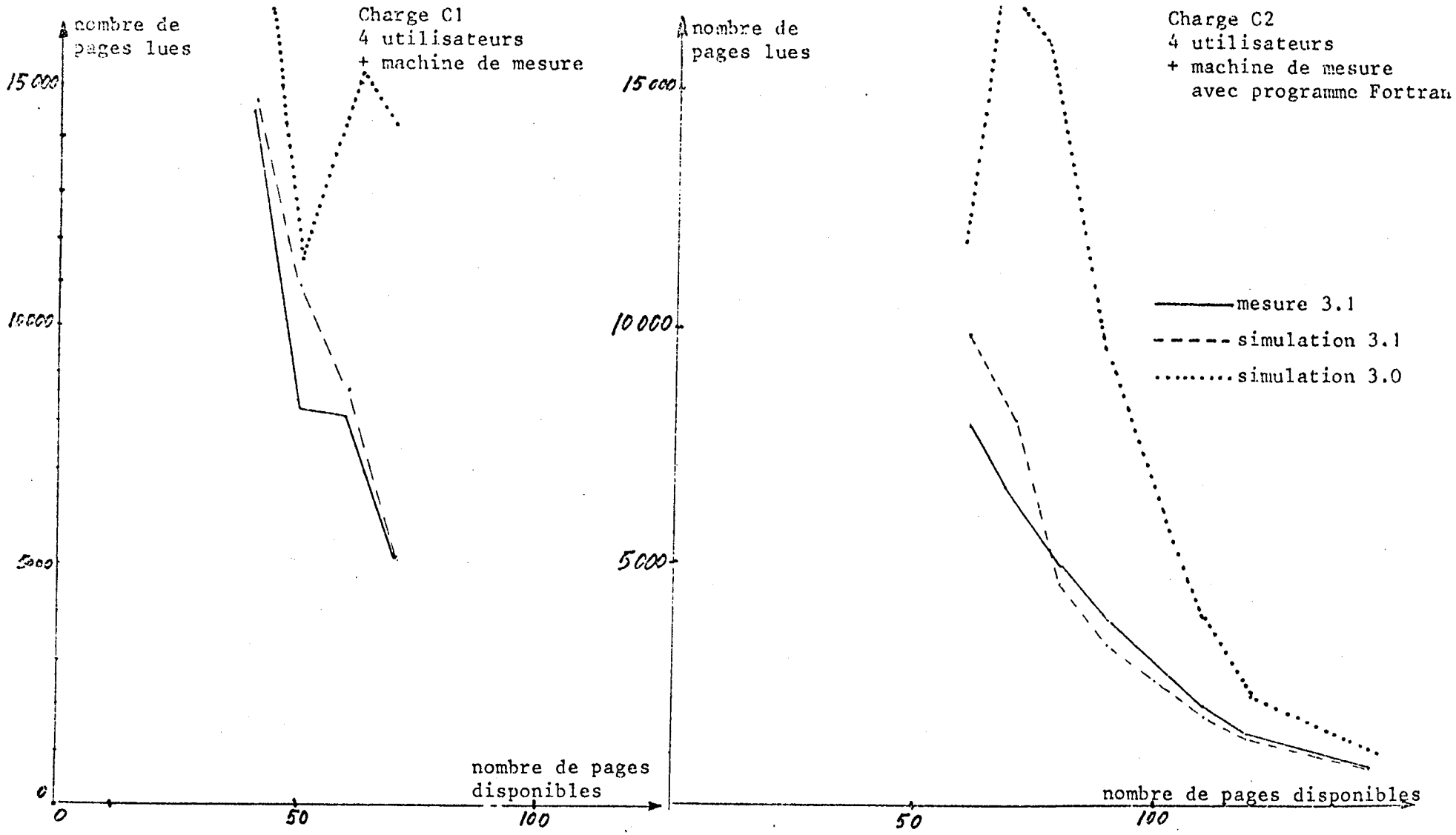
La charge dans ces expériences se compose de quatre machines virtuelles exécutant une série de compilations AED suivies d'assemblages, dont la trace a été utilisée lors de l'expérience mono-utilisateur.

Une trace du programme DUSETIMR, ajustée à son fonctionnement dans les conditions de l'expérience (dépendance du nombre de machines virtuelles notamment), a été utilisée lors de la simulation. Appelons C1 la charge ainsi définie.

La figure 5.3.a. compare le nombre de pages lues dans le système réel et dans les simulations pour différentes tailles de mémoire disponible.

La figure 5.3.b. est du même type pour une charge C2 analogue à la précédente, sauf en ce qui concerne la machine virtuelle de mesures qui active en plus un programme Fortran étalon (lui aussi modélisé) qui dans la réalité sert à des fins d'étude du temps de réponse.

Des différences apparaissent sur ces courbes dues à des séquencements différents de machines virtuelles dans le système réel et dans la simulation. Néanmoins, l'allure générale des courbes est respectée et l'erreur moyenne acceptable (5 % environ).



Figures 5.3.a., 5.3.b. - Validation multi-utilisateurs (pagination) et comparaison des versions 3.0 et 3.1

5.1.3. Validation temporelle avec plusieurs utilisateurs

Le temps pris par les divers algorithmes de CP jouent un rôle important dans l'exactitude temporelle de la simulation. Ces temps fournis par analyse de mesures sur le système réel (cf. § 4.7.) ne sont que des valeurs moyennes. Il importe de savoir si de courtes expériences de simulation utilisant ces données fournissent une précision acceptable.

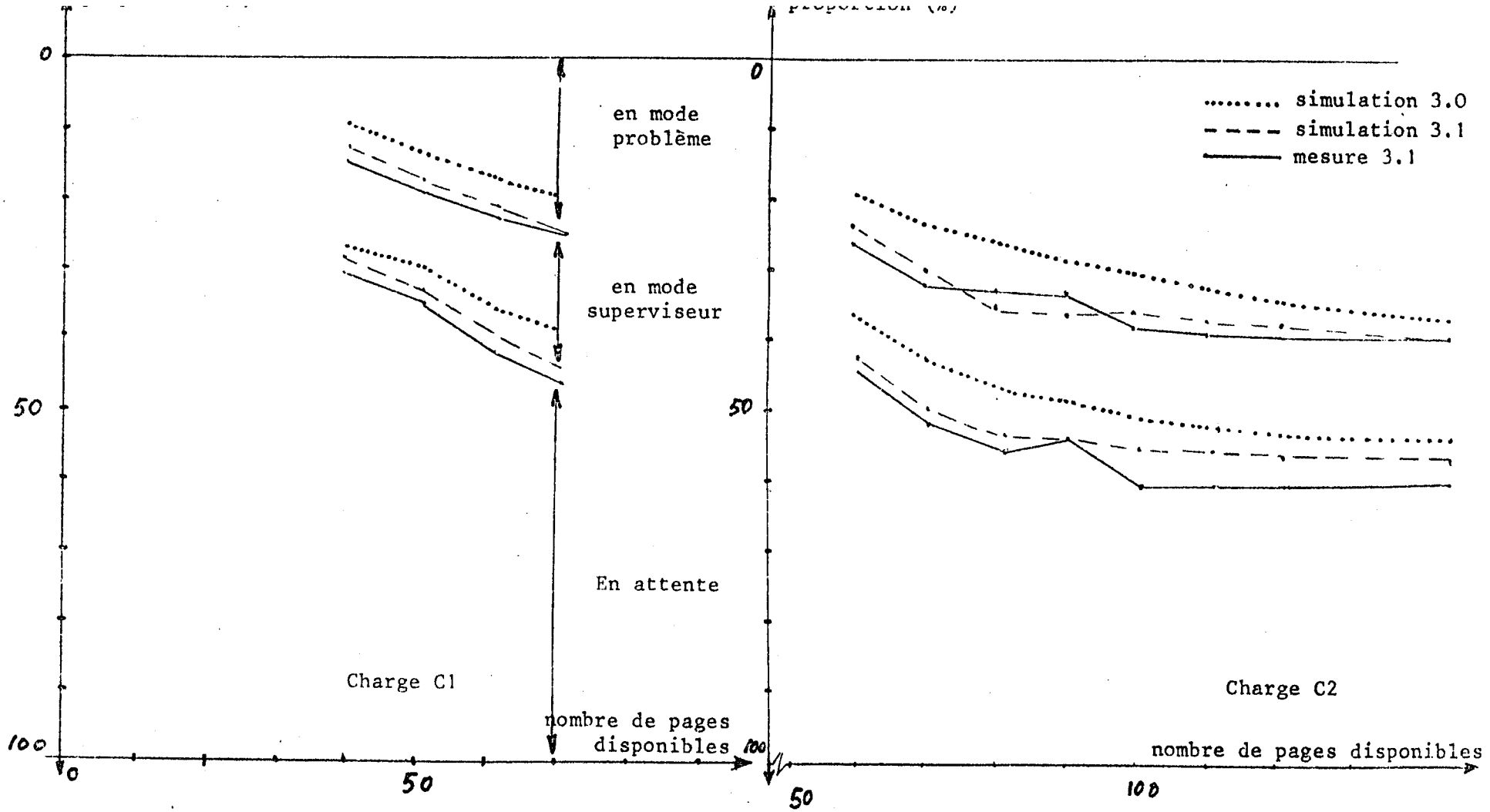
a. Des mesures temporelles sur les expériences multi-utilisateurs précédentes, notamment des taux d'activité de l'unité centrale, du temps mis à servir la première machine virtuelle et de celui à servir la dernière (indicateurs de temps de réponse), ont été prises.

Les figures 5.4. et 5.5. comparent les mesures réelles et les résultats de la simulation. On notera que les temps de réponse ne sont pas connus exactement car la machine de mesures ne peut que constater qu'une des machines virtuelles a terminé son travail entre deux prélèvements. Le point de mesure est ainsi remplacé par un intervalle de temps.

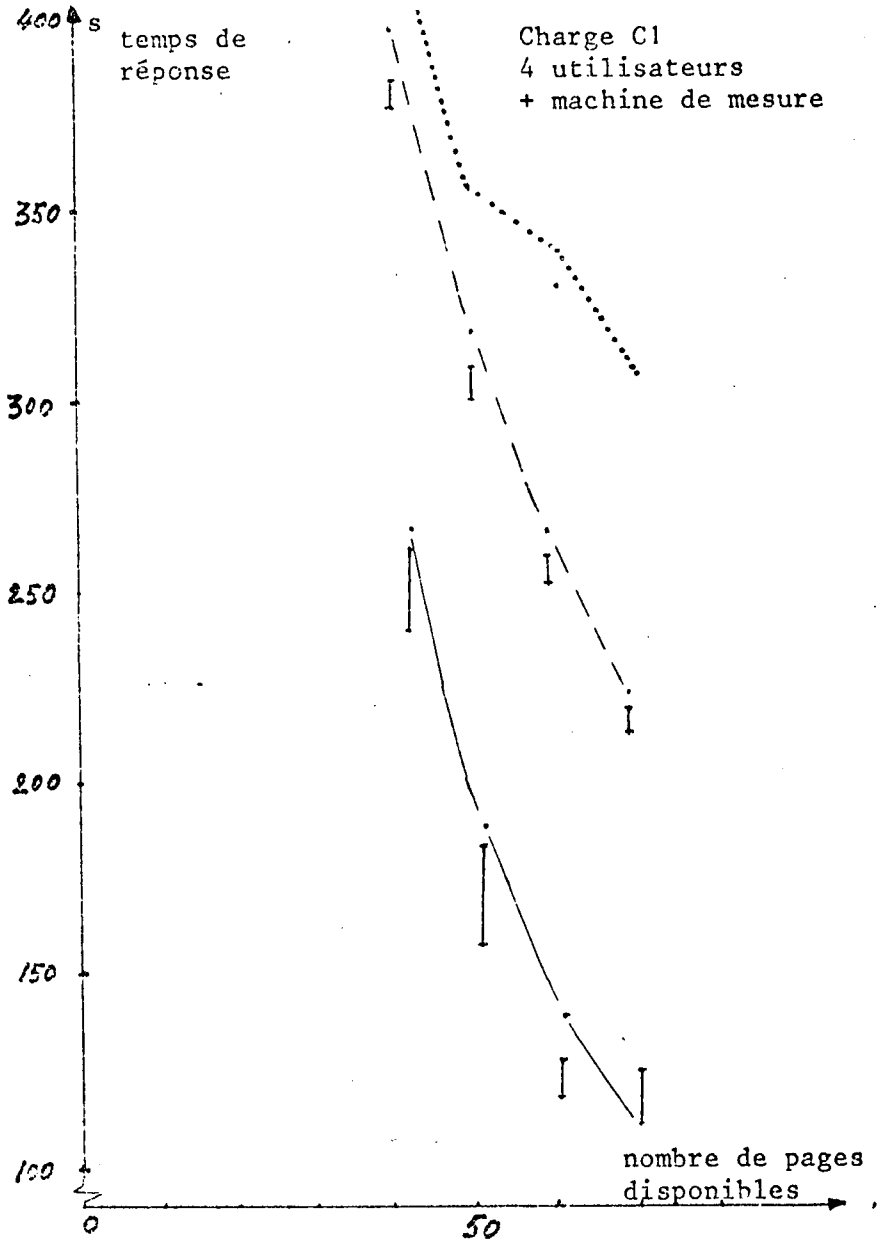
Sauf aux points de mesure de 90 pages, où il semble que le système réel ait eu une anomalie (comportement dû à des variables non prises en compte, ou incident expérimental), la simulation concorde de façon satisfaisante avec la réalité.

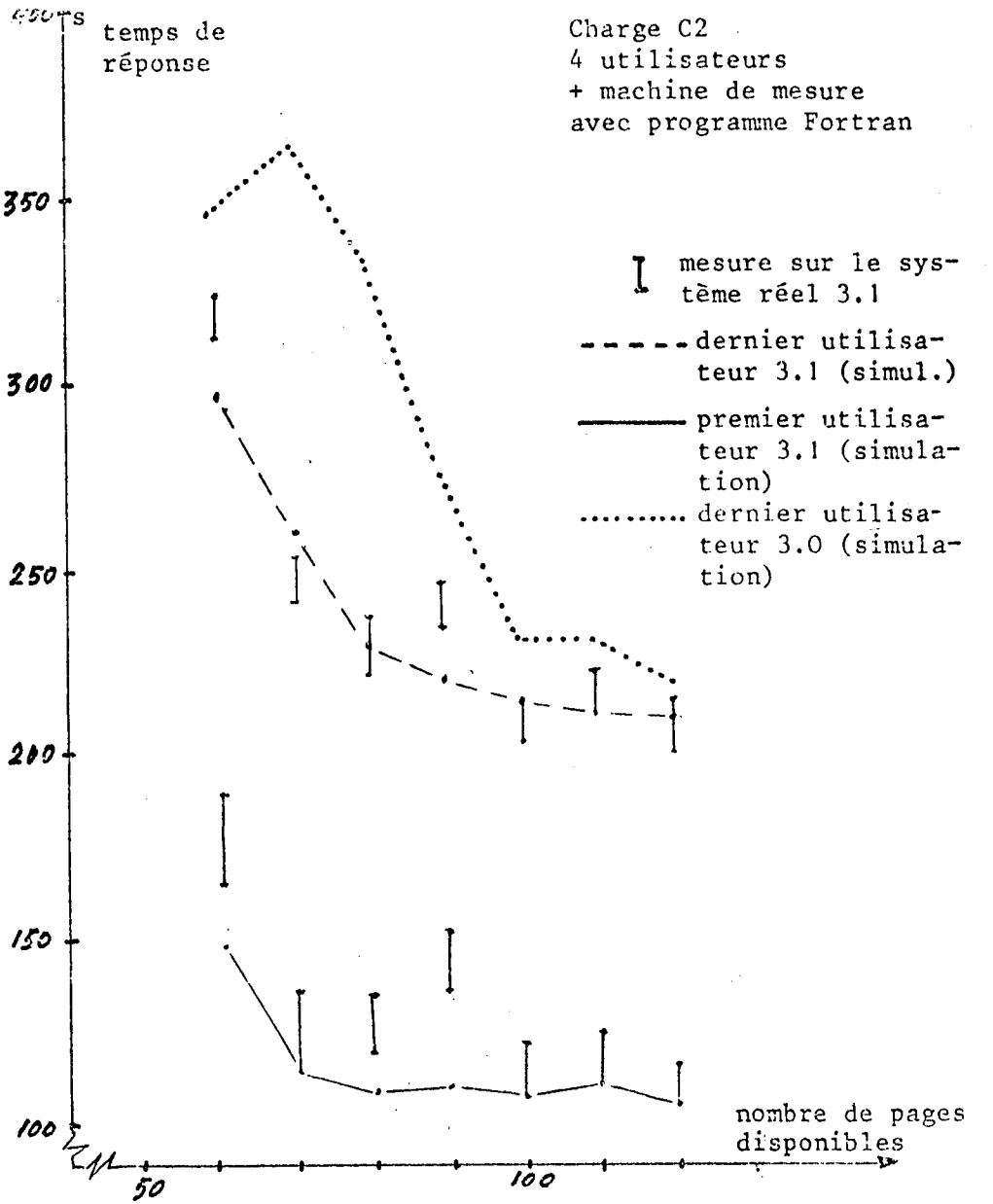
b. Schatzoff et Tillman [75] ont comparé le système réel et le simulateur soumis à une charge de 40 utilisateurs. Ces utilisateurs se servaient en fait de 4 trains de travaux résumés dans le tableau ci-dessous :

numéro de machine virtuelle	train de travaux	temps virtuel UC (s)	nombre d'E/S disque	mise en attente (E/S console)
1 à 16	7 éditions	1.1	84	71
17 à 31	3 assemblages 2 compilations Fortran	9.8	1027	5
32 à 36	7 assemblages	26.2	1308	0
37 à 40	2 assemblages 3 compilations PL/1 2 compilations Fortran	25.3	2671	0



Figures 5.4.a., 5.4.b. - Validation multi-utilisateurs (activité de l'unité centrale) et comparaison des versions 3.0 et 3.1





Figures 5.5.a., 5.5.b. - Validation multi-utilisateurs (temps de réponse) et comparaison des versions 3.0 et 3.1

L'expérience consistait à lancer successivement une machine virtuelle à l'intervalle d'une seconde, afin d'éviter un régime transitoire et à mesurer l'activité du système entre le début de la 40ème seconde et à la fin de la 10ème minute.

Le résultat de 16 expériences de simulation (où certains paramètres du système avaient été modifiés, cf. § 5.3.2.) est résumé dans le tableau ci-dessous :

numéro de l'expérience	proportion de temps en mode problème		proportion de temps en mode superviseur		nombre de pages lues par seconde	
	syst. réel	simulation	syst. réel	simulation	syst. réel	simulation
1	0.286	0.330	0.360	0.313	98.007	89.530
2	0.285	0.286	0.355	0.356	92.675	102.300
3	0.291	0.307	0.335	0.334	87.638	97.700
4	0.283	0.319	0.344	0.334	92.416	95.990
5	0.377	0.427	0.320	0.292	73.844	80.130
6	0.379	0.384	0.299	0.291	66.675	75.250
7	0.365	0.438	0.289	0.292	62.218	76.340
8	0.394	0.415	0.322	0.303	74.176	77.450
9	0.402	0.461	0.250	0.249	43.052	50.000
10	0.398	0.458	0.220	0.216	32.351	37.690
11	0.397	0.469	0.222	0.227	33.607	39.490
12	0.391	0.461	0.249	0.248	42.612	49.670
13	0.417	0.449	0.219	0.212	29.953	35.860
14	0.389	0.475	0.253	0.228	42.775	50.270
15	0.391	0.447	0.247	0.258	40.237	49.750
16	0.449	0.451	0.218	0.215	30.377	37.630

Bien que les valeurs trouvées en simulation dépassent de 10 % celles du système réel quant au nombre de pages lues et à la proportion de temps que le système a passé en mode problème, et soit 4 % en-dessous de la proportion de temps passé par le système en mode superviseur, une analyse factorielle a montré que ces écarts sont comparables à ceux que le système réel peut engendrer :

	proportion de temps en mode problème	proportion de temps en mode superviseur	nombre de pages lues par seconde
système réel (mêmes travaux répétés)	0.0066	0.0034	1.64
simulateur	0.0069	0.0037	1.10

Les causes probables de ces variations sont le "vol de cycles" des entrées-sorties et l'identité du banc de mémoire dans lequel une page est logée qui peuvent faire varier le cycle mémoire jusqu'à 10 % (!).

Par ailleurs, la trace injectée en simulation néglige de prendre en compte des facteurs tels que l'alignement des opérandes pour évaluer la durée de l'instruction.

5.1.4. Calibration du modèle

Compte tenu des précisions obtenues précédemment et du nombre de paramètres énorme que comporte le modèle, aucune calibration fine n'a pu être faite.

Néanmoins, des expériences sur les "robustesse" du simulateur ont été faites. Elles consistaient à effectuer des variations faibles de certains paramètres (38 classés en 8 groupes) et à estimer les effets de ces variations. Ces effets, pour une variation nominale de 30 % pour les sept premiers groupes et 5 % pour le huitième, ne sont pas significatifs au niveau de 5 %. Ainsi, le modèle paraît d'une robustesse suffisante.

L'utilisation de l'analyse factorielle lors de ces expériences s'est révélée fructueuse, compte tenu du très grand nombre de paramètres existants. La description détaillée de celles-ci se trouve dans [75].

5.2. Performances du simulateur

L'activité du simulateur peut se classer dans l'une des trois catégories suivantes :

1) l'initialisation qui construit les structures du modèle et initialise les valeurs d'une manière interactive ;

2) la simulation proprement dite qui fait évoluer le modèle ;

3) l'extraction d'informations, nécessitant des entrées-sorties aussi bien que du calcul, qui peut être modulée de façon interactive.

La dernière catégorie est extrêmement variable puisqu'elle peut aller de la trace de tous les événements intéressant le simulateur (de l'ordre de 1 à 5 ms) jusqu'à l'expression de quelques compteurs en fin d'expérience.

Pour les deux premières catégories, quatre paramètres semblent importants :

. pour la configuration, le nombre de pages physiques du système,

. pour la demande, le nombre d'utilisateurs, le nombre moyen de pages référencées par macro-instruction et le nombre moyen d'instructions agglomérées.

L'effet des deux derniers paramètres dans un cas "vraisemblable" (30 utilisateurs et 150 pages physiques) a été mesuré en introduisant une charge synthétique, égale pour toutes les machines. Cette charge n'introduit pratiquement pas d'attente de l'unité centrale. Comme une telle attente existe dans les expériences proches de la réalité, mais qu'elle ne coûte rien en simulation, nos mesures de l'efficacité du simulateur sont donc pessimistes.

Les figures suivantes illustrent les résultats obtenus. Ces courbes permettent d'apprécier le rapport des temps atteints par des règles de groupage. Ainsi, les programmes normaux (de la charge C1 par exemple) groupés en macros de 10 pages, fournissent un nombre moyen d'instructions supérieur à 2 000.

Les expériences de Schatzoff et Tillman ont utilisé cette valeur de 10 pages afin de rendre le coût de la simulation acceptable.

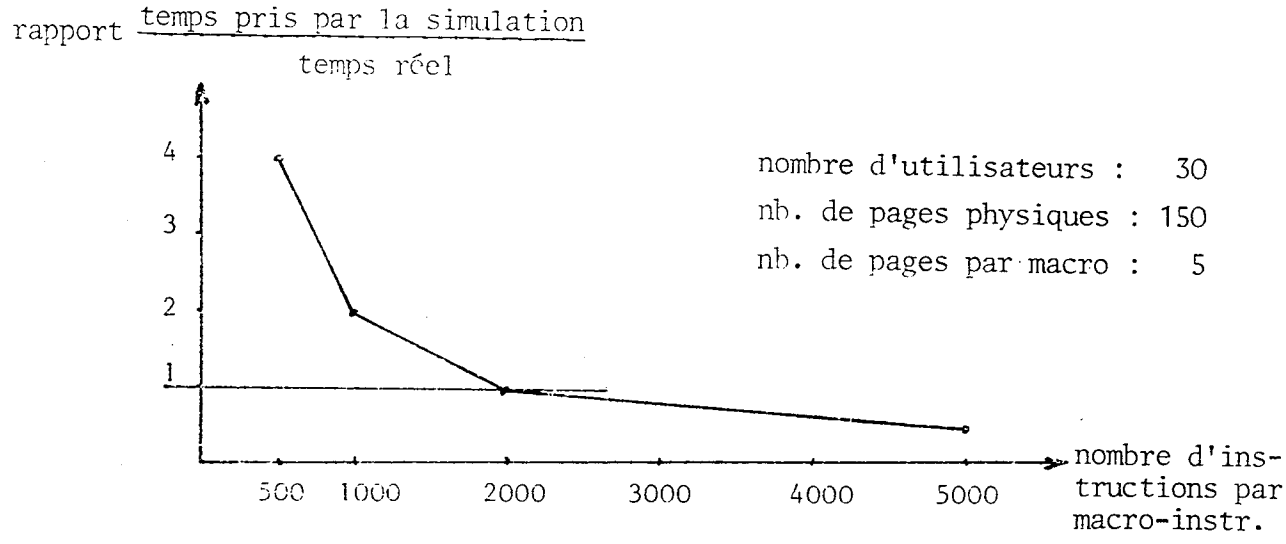


Figure 5.6.a.

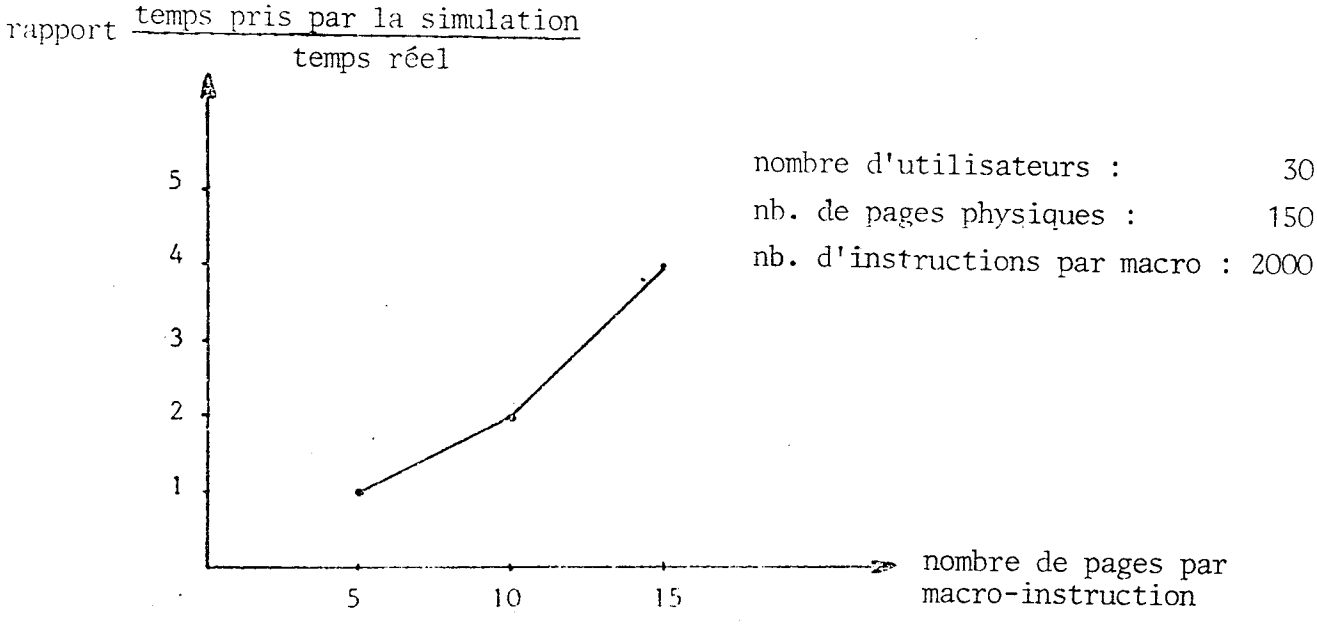


Figure 5.6.b.

5.3. Expériences sur le contrôle de la pagination

5.3.1. Généralités

La pagination est l'un des problèmes des systèmes d'exploitation qui a le plus inspiré les théoriciens. Néanmoins l'apport de ceux-ci sur le plan pratique est resté très limité. Deux raisons principales expliquent ce fait :

1) La notion d'efficacité du système faisant de la pagination a été mal définie. En effet, on a essayé de généraliser (Denning [30] Belady [9]) la formule donnant le taux d'occupation de l'unité centrale lorsqu'une seule tâche est présente : si a est le temps entre défauts de page de cette tâche (supposé indépendant de l'algorithme de remplacement) et b le temps de transfert d'une page (pondéré par un taux fixe de ré-écriture), le taux d'occupation est : $E = \frac{a}{a + b}$. Si la pagination se fait sur une seule unité, lorsqu'on a n tâches différentes ce taux d'activité n'est pas

$\sum_{i=1}^{n} \frac{a_i}{a_i+b_i}$ qui peut d'ailleurs être supérieur à 1,

mais $\frac{\Sigma a_i}{\max(\Sigma a_i, \Sigma b_i)}$ dans le cas général.

Rousset [73] a montré que les exceptions correspondent seulement aux cas où il existe sur la ressource la moins saturée une tâche qui force la ressource saturée à attendre.

Par exemple si $\sum_{i=1}^{n} a_i > \sum_{i=1}^{n} b_i$, où l'unité centrale est saturée, et

j tel que $b_j > \sum_{\substack{i=1 \\ i\neq j}}^{n} a_i$ ou $a_j + b_j > \sum_{i=1}^{n} a_i$. Le taux d'activité est alors

$\frac{\Sigma a_i}{a_j+b_j} > 1$. C'est bien le cas lorsqu'il n'y a qu'une seule ressource.

L'expression générale $\frac{\Sigma a_i}{\max(\Sigma a_i, \Sigma b_i)} = \min(1, \frac{\Sigma a_i}{\Sigma b_i})$ valable dans le cas déterministe, où le modèle est simplifié, permet néanmoins une bonne explication des phénomènes "d'écroulement" du système ou d'allocation optimale de mémoire (appendice 3).

Bien que l'expression du taux d'activité se complique lorsqu'on considère des variables aléatoires, le cas simple peut nous servir de référence.

2) Le modèle du "Working Set" pour le comportement de programmes (Denning [22]) a le mérite d'être indépendant du système, notamment de l'algorithme de remplacement, mais reste difficile à réaliser pratiquement. En effet, chaque intervalle de temps, ou nombre d'instructions, détermine un "working set" du programme. Si on l'accepte comme prédicteur et que l'on veuille conserver les pages correspondantes en mémoire, on reporte le problème d'efficacité sur le choix de l'intervalle de temps. Mais si l'on est amené à faire varier cet intervalle dynamiquement, le problème d'allocation de la mémoire devient crucial.

Il paraît plus simple d'utiliser les fonctions "durée de vie" (Belady [9]) pour lesquelles la variation de l'espace de travail d'un programme peut se faire graduellement au rythme des défauts de page.

D'une façon générale, le contrôle de la pagination s'effectue par deux actions complémentaires :

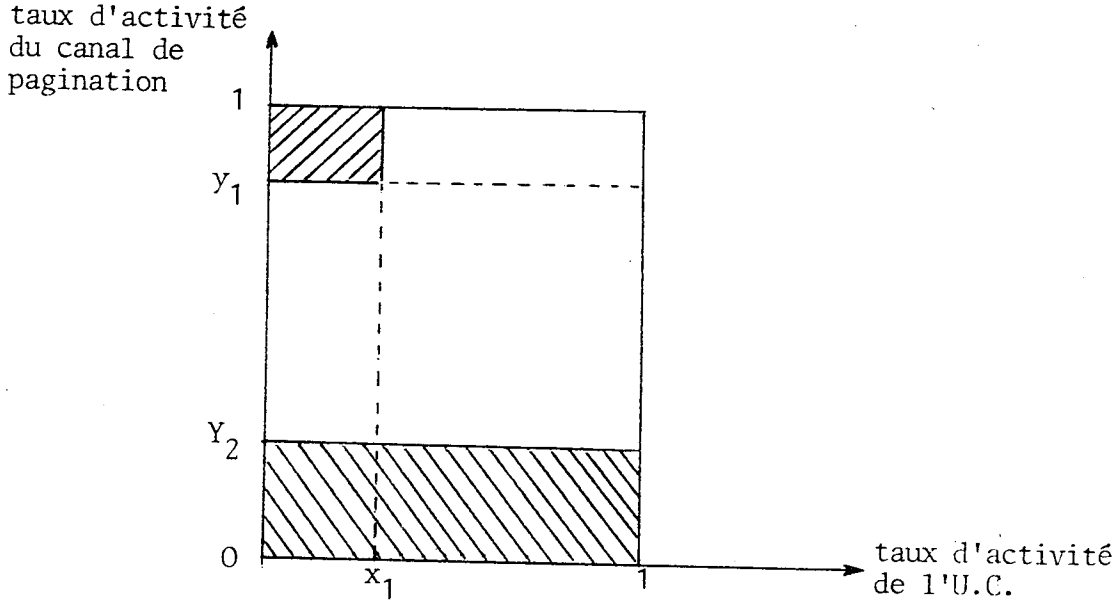
i) l'une à "long terme" décidant l'introduction ou l'évacuation d'une tâche en mémoire centrale, car cette action peut être coûteuse par ses conséquences : un programme commençant à s'exécuter sans page en mémoire fera un certain nombre de défauts de page très rapprochés avant d'atteindre un taux raisonnable.

Les raisons nécessitant l'évacuation peuvent être la fin d'un programme, une entrée-sortie vers des consoles (dont la durée est de l'ordre de 10 secondes), ou l'estimation qu'il y a trop de programmes en mémoire centrale.

Les raisons nécessitant l'introduction d'un programme peuvent être l'estimation que le nombre de défauts de page résultant de cette introduction sera acceptable (immédiatement après l'évacuation d'un autre programme, par exemple), ou la nécessité temporelle de la faire (tâche avec date limite, par exemple).

Historiquement, la nécessité de limiter l'accès à la mémoire centrale n'est pas apparue immédiatement. Le phénomène correspond à un partage entre de trop nombreux programmes en mémoire centrale, qui produisait de nombreux défauts de page (tambour saturé) alors que l'unité centrale est inactive, a été baptisé "thrashing".

Un remède simple apporté à ce phénomène consiste à limiter la charge ("load leveler") avec asservissement. On mesure périodiquement le taux d'occupation de l'unité centrale et du canal de pagination, en ayant défini deux régions critiques : l'une où le taux de pagination excède y_1 et le taux d'utilisation de l'unité centrale est inférieur à x_1, l'autre où le taux de pagination est inférieur à y_2. La stratégie adoptée consiste simplement à évacuer un programme si le point de fonctionnement est dans la première région et à en introduire un nouveau si ce point est dans la seconde région.



"Load leveler" du M44/44X, ancêtre de CP
(IBM report, Shils 1968)

Cet asservissement primitif a donné des résultats acceptables. Néanmoins, dans le souci d'éviter autant que possible les évacuations de programme, on a essayé d'apporter plus de soin à l'introduction des programmes.

Le moyen le plus simple consiste à fixer un nombre maximum de programmes en mémoire, n. Si s_0 est l'espace qu'il faut à un programme moyen pour que son temps entre défauts de page $e(s_0)$ soit égal au temps de transfert T, on pourra prendre n égal à entier $(\frac{M}{s_0})$. Le système CP 67 jusqu'à la version 3.0 a appliqué cette stratégie. Nous avons mis en évidence la rapidité avec laquelle le "thrashing" peut survenir dans ce cas au § 5.3.2.

Un moyen plus complexe consiste à estimer les demandes futures des tâches afin de déterminer si l'évacuation est nécessaire ou si l'introduction d'une tâche est possible. La version 3.2 de CP 67 applique ce principe d'une façon complexe [75b]. Auparavant, Rodriguez et Dupuy se basant sur la notion de working-set, avaient aussi introduit l'estimation dynamique du besoin en mémoire des programmes [71].

Ces estimations introduisent des relations entre la stratégie à long terme et les décisions qu'il faut prendre à court terme.

Notons que des modèles simplifiés ont étudié la détermination puis la régulation du "degré de multiprogrammation" en simplifiant les phénomènes à court terme où l'on suppose les programmes semblables ([18], [6], [19], [27]).

Nous proposons une stratégie qui en pratique s'applique aussi bien au long terme qu'au court terme au § 5.3.3.

ii) La seconde action à court terme, qu'on appelle algorithme de remplacement, décide, lors d'un défaut de page, à quelle tâche sera enlevée la page et laquelle.

Ce choix s'est initialement fait sans tenir compte de l'identité de la tâche de la manière suivante :

FIFO : dans l'ordre de leur entrée en service ; il suffit donc d'un index circulaire pour choisir la page. Malheureusement, cette méthode présente des inconvénients, notamment de ne pas assurer qu'un espace plus grand produirait moins de défauts de page [10].

RANDOM : au hasard, en général de façon équirépartie sur toutes les pages possibles. Cet algorithme demande l'usage d'un générateur de nombre aléatoire, qui est plus coûteux que le précédent pour un résultat qui ne peut être très bon par nature.

LRU : d'après l'ordre inverse de leur dernière utilisation. Cet ordre, bon prédicteur de l'usage futur des pages d'un programme, demande du matériel supplémentaire pour tenir à jour les utilisations au fur et à mesure des références. Il a surtout été utilisé dans le cache ([56], [11]) ou pour des niveaux de mémoire gérés par programme (pagination du tambour sur disque de MULTICS [74]). On ne sait s'il s'appliquerait efficacement au cas de la mémoire centrale sans tenir compte de l'identité des tâches.

L'hypothèse de l'existence d'une fonction de durée de vie implique que les algorithmes de remplacement employés dans l'espace d'un programme n'ont pas d'influence sur le nombre de défauts de page. Pour des algorithmes qui essaient d'approcher le LRU, cette hypothèse semble valable. Au § 5.3.4. nous décrivons l'algorithme de CP 67 version 3.1 qui essaie de tenir compte de l'usage récent et nous décrivons les expériences de comparaison des versions 3.0 et 3.1.

Le choix préalable de la tâche à qui l'on va prendre une page semble nécessaire à toute politique qui s'appuie explicitement sur le comportement des programmes, notamment pour la gestion à long terme. Ce souci apparaît même dans la façon dont les algorithmes de remplacement réels ont modifié les algorithmes précédemment décrits.

L'algorithme décrit au § 5.3.3. propose une gestion simple basée sur la recherche du taux maximum d'occupation de l'unité centrale à degré de multiprogrammation constant.

5.3.2. Limite statique du degré de multiprogrammation

Dans le but d'éviter l'écroulement du système, la version 3.0 de CP 67 applique une politique simple : un maximum est fixé pour chacune des deux sous-classes des travaux interactifs et des travaux de fond.

Les courbes de taux d'utilisation de l'unité centrale en fonction du nombre de machines virtuelles décroît d'une façon assez lente avec le nombre d'utilisateurs (figures 5.8.a. et 5.9.a.). Ce phénomène, qu'il ne faut pas attribuer à une impossibilité de "thrashing", tient au choix des nombres limites (9 + 6 pour 200 pages, 8 + 4 pour 150 pages par interpolation, cf. § 4.3.2.) qui ne sont pas trop éloignés de ceux trouvés expérimentalement par Rodriguez [70] ou théoriquement sur des modèles de réseaux de serveurs exponentiels (Baixas [6], Brandwajn [18]).

Afin de montrer l'effet d'un mauvais choix, nous avons étudié, pour une configuration donnée et une charge donnée, la variation induite par les limites du degré de multiprogrammation.

Chaque machine virtuelle exécute une commande synthétique définie d'après les mesures de Leroudier [55]. Cette commande synthétique a la forme suivante :

initialisation : 30 e/s sur disque
(dont 15 sur le disque système et 15 sur le disque virtuel de l'utilisateur)

travail : 7 fois 10^5 instructions sur 15 pages
10^5 instructions sur 15 pages dont 5 des précédentes
x instructions privilégiées
1 entrée-sortie console (7 s)

fin : 9 e/s disque (sur disque virtuel de l'utilisateur)

Le temps d'exécution d'une instruction est de 1.5 microsecondes, ce qui donne un temps d'unité centrale de 2.1 secondes. Appelons C'_1 la commande où x = 40, soit une instruction privilégiée pour 5000 instructions normales, et C'_2 la commande où x = 160, soit une instruction privilégiée pour 1250 instructions normales, qui semble plus proche des commandes réelles de courte durée.

Lorsqu'on fixe le nombre d'utilisateurs, c'est-à-dire de machines virtuelles, chacun reçoit un numéro. Chaque disque virtuel se compose de 7 cylindres consécutifs sur un disque. On a donc 28 utilisateurs par disque IBM 2314 de 200 cylindres.

On a simulé, sauf mention explicite du contraire, un seul canal sélecteur d'entrée-sortie sur disques, avec un disque système et le nombre nécessaire de disques utilisateurs.

Les commandes utilisées sont décrites sous la forme de macro-instructions définies au § 4.2.2. Un programme de machine virtuelle comprend une liste de telles macro-instructions et des instructions d'itération (boucles "pour" imbriquées). Un interpréteur agissant en co-routine vis à vis du simulateur (cf. § 3.5.) fournit à celui-ci, sur sa demande, la prochaine macro-instruction qui sera exécutée par la $i^{ème}$ machine virtuelle. Cet interpréteur peut travailler sur des programmes "réentrants" le contexte d'un utilisateur étant la pile des valeurs d'itération et le compteur compteur ordinal.

L'enregistrement de ces programmes se fait soit dans un module qui est compilé, où les valeurs par défaut sont définies, soit dynamiquement à l'initialisation d'une expérience. Pour ces expériences, nous avons opéré sur la version 3.0 de CP 67, avec 150 pages disponibles, un canal d'entrée-sortie et sous la charge de 30 utilisateurs demandant des commandes C'_2.

Les figures 5.7. résument les résultats obtenus. L'écroulement du système apparaît nettement lorsque le degré maximum de multiprogrammation dépasse 17, alors que la valeur normale du système est 12 : le taux d'activité de l'unité centrale baisse brutalement alors que le canal de pagination est quasi saturé.

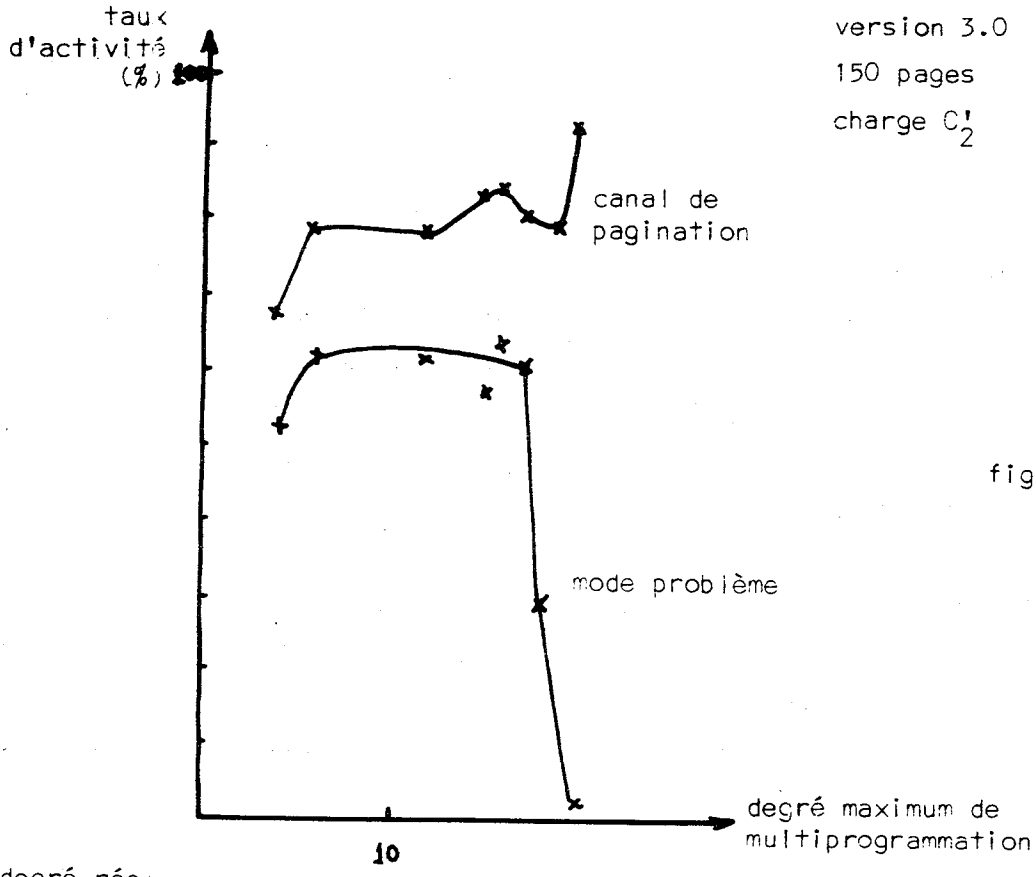


figure 5.7.a.

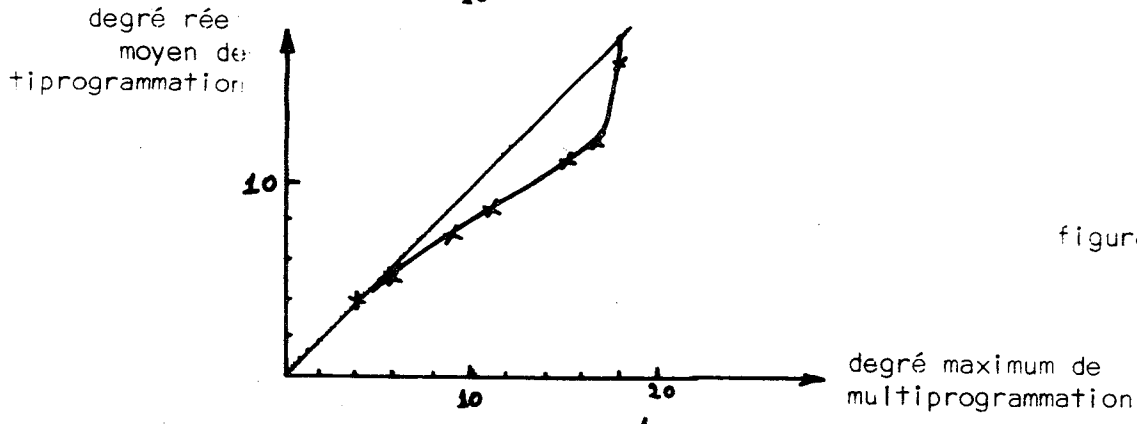


figure 5.7.b.

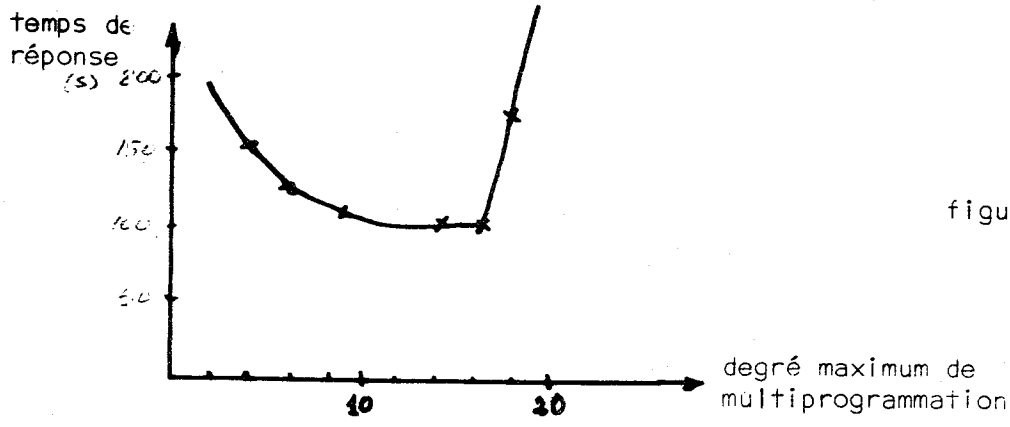


figure 5.7.c.

Figures 5.7.a., 5.7.b., 5.7.c. Influence du degré maximum de multiprogrammation

Notons que l'écroulement paraît plus brutal que les expériences réelles qui ont pu être observées. Le fait d'avoir des demandes semblables et déterministes, et surtout des macro-instructions agglomérant 15 pages, rend ce problème plus dramatique.

On note aussi que la courbe du degré réel moyen par rapport au degré maximum a une forme caractéristique : lorsque ce dernier est trop bas (ici $\leqslant 4$), le degré réel lui est égal, car la charge est suffisante pour alimenter continuellement le degré réel.

: lorsque le degré maximum est trop haut (ici $\geqslant 17$), le degré réel lui est égal, car l'avancement des travaux ne se fait pratiquement pas et les programmes se volent mutuellement des pages avec un degré réel de multiprogrammation égal au degré maximum.

Entre ces deux zones, le degré réel est inférieur au degré maximum à cause du manque de synchronisation des demandes des programmes : à certains instants, les programmes sont en entrée-sortie console, même s'ils avaient pu alimenter un degré de multiprogrammation supérieur, en moyenne.

5.3.3. Algorithme adaptatif de remplacement et d'admission

L'étude simplifiée du partage de la mémoire a été faite en annexe 3 afin de maximiser le taux d'occupation de l'unité centrale lorsque les tâches multiprogrammées sont caractérisées par leur temps moyen entre défauts de page, supposé fonction seulement de l'espace mémoire (fonction "durée de vie"). En supposant que ces fonctions possèdent des dérivées discontinues en un nombre fini de points, donc des dérivées à droite et à gauche, nous savons qu'à l'optimum nous devons avoir :

(1) $e_i'^{+}(s_i) \leqslant e_j'^{-}(s_j) \quad \forall i,j = 1,..N \text{ et } i \neq j$

L'algorithme de remplacement que nous proposons essaie de se rapprocher d'un optimum (éventuellement local) de la manière suivante :

si la tâche i fait un défaut de page, on regarde si la condition 1 est vérifiée pour tout $j \neq i$. Si oui, on prendra la page à la tâche i elle-même, sinon on prendra la page à une tâche j telle que $e'_j(-s_j) < e'_i(+s_i)$, en particulier on peut donc prendre j tel que $e'_{j_0}(-s_{j_0}) = \min_{j \neq i}(e'_j(-s_j))$.

En pratique, ces dérivées seront estimées d'après des accroissements de $e_j(s)$ lorsque s prend des valeurs entières. Plus précisément, lorsque la tâche i fait un défaut de page, on calcul $\min_{j \neq i}(e_j(s_j)-e_j(s_j-1))$.

Si cette valeur, atteinte pour $j = j_0$, est inférieure à $e_i(s_i+1) - e_i(s_i)$, on retire une page de la tâche j_0 pour l'attribuer à la tâche i,

sinon on remplace une page de la tâche i elle-même par la page manquante.

Les divers problèmes attachés à cette méthode sont les suivants :

i) dans quelles conditions le problème mathématique continu est-il résolu par l'algorithme itératif ?

ii) dans quelles conditions la discrétisation conserve-t-elle un sens aux solutions de i) ?

iii) dans quelle mesure la réalité correspond-elle aux problèmes mathématiques précédents ?

Les deux premiers problèmes, mathématiques, laissent une certaine latitude aux théoriciens pour échafauder les hypothèses nécessaires : par exemple, si l'optimum local est global, en particulier avec des fonctions convexes ou concaves, et si la règle de distribution de contrôle sert tour à tour toutes les tâches, l'algorithme itératif tend vers l'optimum. Mais ils ne représentent que la partie relativement bien définie de la question.

Pour l'utilisation pratique, il faut absolument aborder le troisième problème et il semble vain de préciser trop les solutions des deux premiers, hors l'intérêt mathématique, par rapport aux conclusions du troisième problème.

Ici, ce problème correspond à savoir si la fonction durée de vie a un sens, quelle mesure ou estimation peut en être faite et si les techniques appliquées permettent dans la réalité de réduire le thrashing comme elles le feraient vis à vis du modèle mathématique ?

Le simulateur de CP 67 permet de substituer un modèle à la réalité pour effectuer ces comparaisons. La mesure du temps entre défauts de page montre qu'il faut plutôt considérer ce temps comme une variable aléatoire. Alors sa variance mesurée semble grande. En considérant la moyenne arithmétique de n mesures successives (n étant compris entre 4 et 9), d'une part on diminue sensiblement la variance (divisée par n) et, d'autre part on espace dans le temps d'éventuelles décisions contradictoires vis à vis d'une tâche : en effet, on ne cherchera pas à prendre en compte une tâche qui vient de recevoir une page supplémentaire avant qu'elle ait fait n défauts de page.

Cette méthode ne rend pas compte de la variation, avec le temps, au cours de l'exécution d'une tâche,de cette valeur. En fait, on peut pallier à ce défaut en utilisant un lissage exponentiel sans biais :

$$v_m = \alpha e_m + (1 - \alpha) v_{m-1} \quad \text{avec} \quad 0 \leqslant \alpha \leqslant 1 \text{ et } e_0 = v_0$$

où v_m, la nouvelle estimation est une pondération linéaire de l'ancienne v_{m-1}, et l'observation (qui en regroupe déjà n, ici) e_m.
Si α est proche de 1, on accorde plus de poids aux informations récentes qu'aux informations passées, tandis que α proche de 0 fait l'inverse.

Les programmes synthétiques que nous avons essayés sont peu sensibles à cette déformation dans le temps (ce sont principalement des boucles). Un bon choix de α dépend de la valeur n considérée,qui intègre déjà le passé, de la nature des programmes et de l'espace moyen accordé.

Notons que l'effet des macro-instructions de regrouper les défauts de page, se combine parfaitement avec une moyenne de temps entre ces défauts prise sur n défauts successifs.

De même, pour un programme ayant s_i pages en mémoire centrale, on connaît en général soit $e_i(s_i+1)$, soit $e_i(s_i-1)$, suivant qu'on lui a récemment diminué son espace de travail ou qu'on l'ait augmenté d'une page.

Ainsi si l'on connaît $e_i(s_i)$ et $e_i(s_i+\varepsilon)$, avec $\varepsilon = \pm 1$, on estime $e_i(s_i-\varepsilon)$ par extrapolation linéaire :

$$e_i(s_i-\varepsilon) = 2e_i(s_i) - e_i(s_i+\varepsilon)$$

calculé au moment où l'algorithme de remplacement en a besoin.

L'admission (ou l'éviction) d'un utilisateur en (ou de la) mémoire centrale, se fait au vu de la comparaison entre la charge sur l'unité centrale et celle du canal de pagination.

Supposons qu'il y ait k tâches en mémoire centrale et que

$r = \dfrac{\sum_{i=1}^{k} e_i(s_i)}{kT}$ est inférieur à $\beta- < 1$, on dit qu'il y a significativement début de "thrashing" et un usager tel que $e_i(s_i)$ est le plus petit, est évincé. Si r est supérieur à $\beta+ > 1$, on dit que le partage peut être augmenté et l'on introduit une nouvelle tâche.

Nous avons pris $\beta- = 0.8$ et $\beta+ = 1.2$ afin de ne pas trop osciller entre l'insertion et l'éviction d'usagers. En pratique, si les bornes statiques ne semblent pas convenables, on peut les asservir à l'activité réelle de pagination.

Ainsi essayé (cf. figure 5.8.a.) cet algorithme donnait des résultats inférieurs à l'algorithme réel (version 3.0) sur la charge C_2'. L'analyse des simulations a montré que le phénomène suivant arrivait : l'extrapolation linéaire conduit à attribuer une page supplémentaire à un programme pouvant déjà accueillir les pages nécessaires à l'interaction. Cette page n'est pas rendue par suite de la temporisation qui attend n défauts de page pour valider une seule mesure, alors que ces défauts de page se trouvent très espacés ou n'arrivent pas avant l'interaction.

Pour pallier à ce défaut, on n'effectue la temporisation lors d'un accroissement d'espace que :

. soit si la valeur estimée est inférieure à un maximum (nous avons pris deux fois le temps de transfert moyen d'une page, soit 24 ms),

. soit si le premier des défauts de page constaté dépasse la valeur estimée par l'extrapolation.

Avec cette contrainte sur la temporisation, on voit que l'algorithme de remplacement modifié est légèrement meilleur que l'algorithme de la version 3.0, avec la charge C_2'.

Lorsqu'on dédouble les macro-instructions en utilisant deux fois plus de pages pour le même nombre d'instructions, ce qui donne la charge C_4', on voit que la version 3.0, avec le niveau standard de multiprogrammation, provoque un certain écroulement du système, alors que les deux modifications que nous proposons résistent fort bien à ces demandes (figures 5.8.a. et 5.8.b).

Avant de pouvoir conclure définitivement, il faudrait expérimenter dans le mode réel afin de voir si le réglage des paramètres n, α, $\beta-$, $\beta+$, s'avère délicat. Au cas où ils ne le seraient pas, on pourrait affirmer que la répartition d'espace dans un système paginé peut se faire simplement.

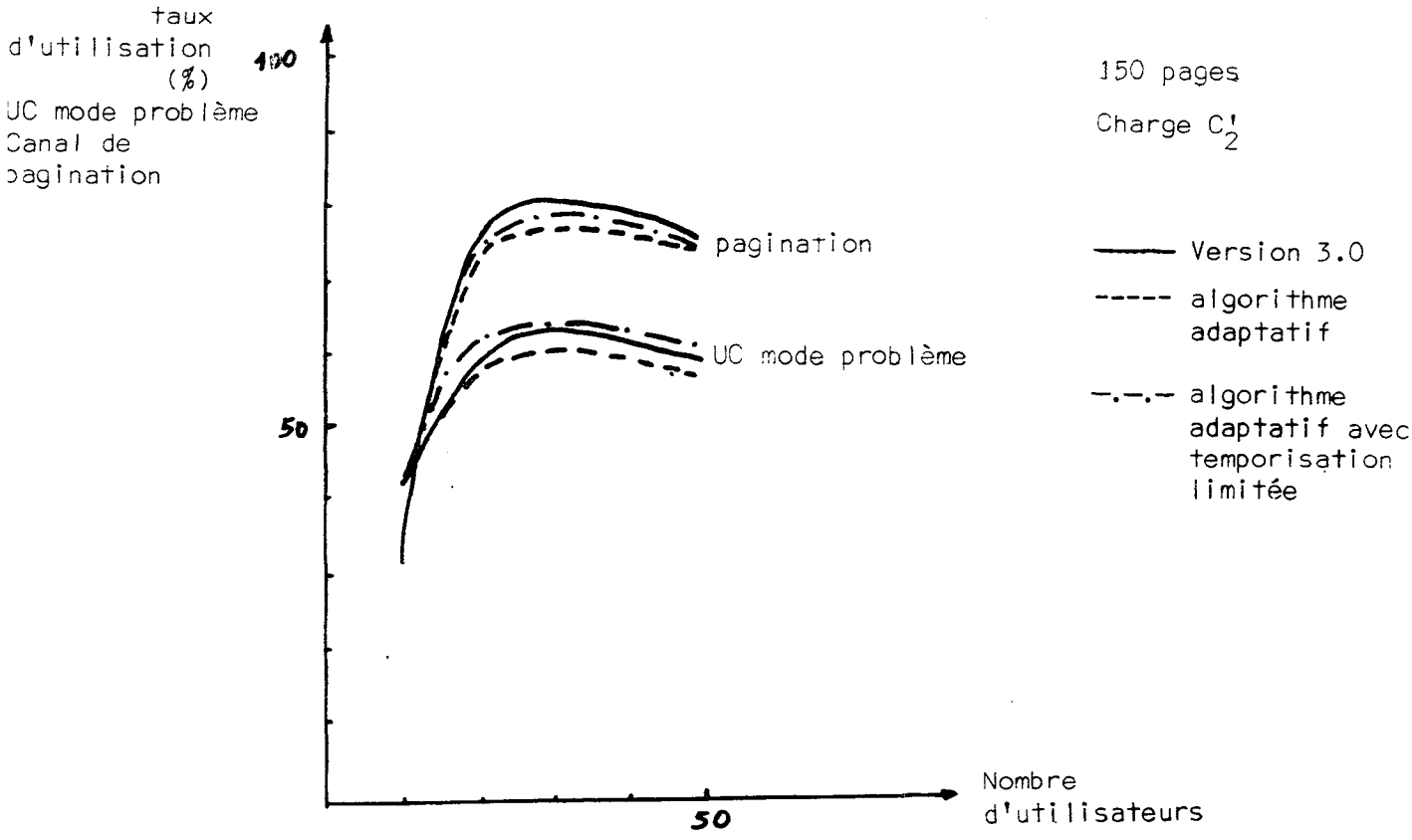


Figure 8.a.

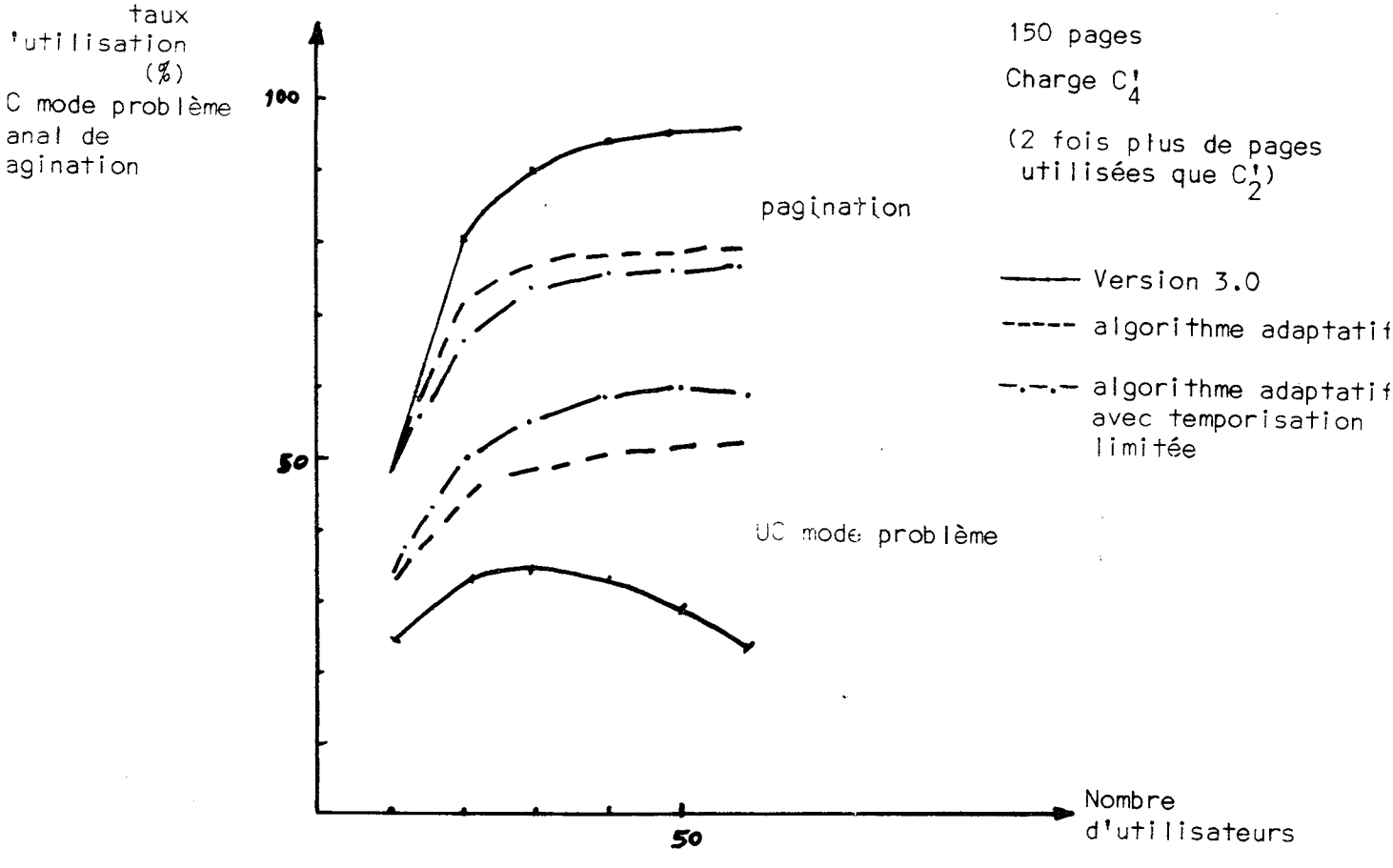


Figure 5.8.b.

Figures 5.8.a., 5.8.b. Essai des algorithmes adaptatifs avec ou sans

5.3.4. Comparaison des versions successives de CP

Il s'agit des versions 3.0 et 3.1 de CP qui diffèrent par certains services, mais surtout par le souci d'une plus grande efficacité. L'originalité de l'utilisation du simulateur provient du fait que l'expérimentation directe à Cambridge (Mass.) sur la version 3.0 n'était plus possible .. car la version suivante était en service. Précisons à nouveau les détails des algorithmes de remplacement utilisés.

La version 3.0 opère ainsi pour trouver une page à remplacer : on balaie, à partir de la valeur d'un index courant, la table des pages présentes en mémoire et l'on choisit :

i) la première page libre, s'il y en a une,

ii) sinon la première page appartenant à un utilisateur "inactif", s'il y en a une,

iii) sinon la première page non bloquée d'un utilisateur actif ("vol de pages").

Lors de la compétition (iii), ce mécanisme est principalement FIFO.

La version 3.1 sur laquelle la validation a porté, prend en compte les indicateurs de référence et d'usage de page de la manière suivante :

ii) sinon une page ayant ces deux indicateurs à zéro (page non référencée)

iii) sinon la première page non bloquée.

Au cours de ce balayage, les indicateurs de référence sont mis à zéro.

Ainsi, l'usage récent (depuis la dernière interruption pour page manquante) se substitue à l'effet FIFO pur.

Des expériences de simulation sous les charges C1 et C2 avec le modèle de l'algorithme de remplacement version 3.0, ont été faites. Le résultat de ces expériences comparé aux mesures (figures 5.3 à 5.5) montre que l'algorithme de la version 3.1 semble meilleur que celui de la version 3.0 car le taux de pagination a diminué et la proportion de temps en mode problème augmenté (5 à 40 % en valeur relative). Compte tenu de l'incertitude observée lors de la validation, cette amélioration est certaine. Elle se compare à celle que des modifications plus profondes ont obtenu sur cette version 3.0.

De plus, la version 3.0 présente des anomalies de fonctionnement (figure 5.3.) (plus de défauts de page lorsqu'on accroît l'espace) identifiées précédemment [10] alors que la version 3.1 n'en présente plus.

5.4. Autres expériences de simulation de CP

5.4.1. Récupération de l'inactivité de l'unité centrale

Fréquemment on entend dire que les périodes d'inactivité de l'unité centrale pourraient être utilisées à faire des mesures, des tests sur divers organes, de la recopie.

Comme j'espère l'avoir démontré dans le chapitre 2, si ces tâches présentent une variation statistique dans leur durée, elles induiront toujours une certaine attente sur l'unité centrale. En d'autres termes, on ne pourra alors récupérer toute l'attente.

Une série d'expériences confirme ce fait : au lieu d'introduire ces tâches supplémentaires, compatibles avec l'attente que subit l'unité centrale, on les a définies par le taux d'instructions privilégiées présentes dans les tâches des utilisateurs, car chacune de ces instructions privilégiées nécessite une "simulation" de la part de CP, donc impose un travail supplémentaire.

Les figures 5.9. comparent les taux d'utilisation avec les charges C'_1 et C'_2 définies au § 5.3.2. correspondant respectivement à un instruction privilégiée pour 5000 instructions normales et un instruction privilégiée pour 1250 instructions normales. Le système est constitué de la version 3.0 avec 200 pages disponibles et un niveau maximum de multiprogrammation de 15 (9 interactifs + 6 travaux de fond).

On voit que le taux d'utilisation global de l'unité centrale et des canaux de pagination restent semblables, avec un maximum vers 30 utilisateurs puis une décroissance assez lente au-delà.

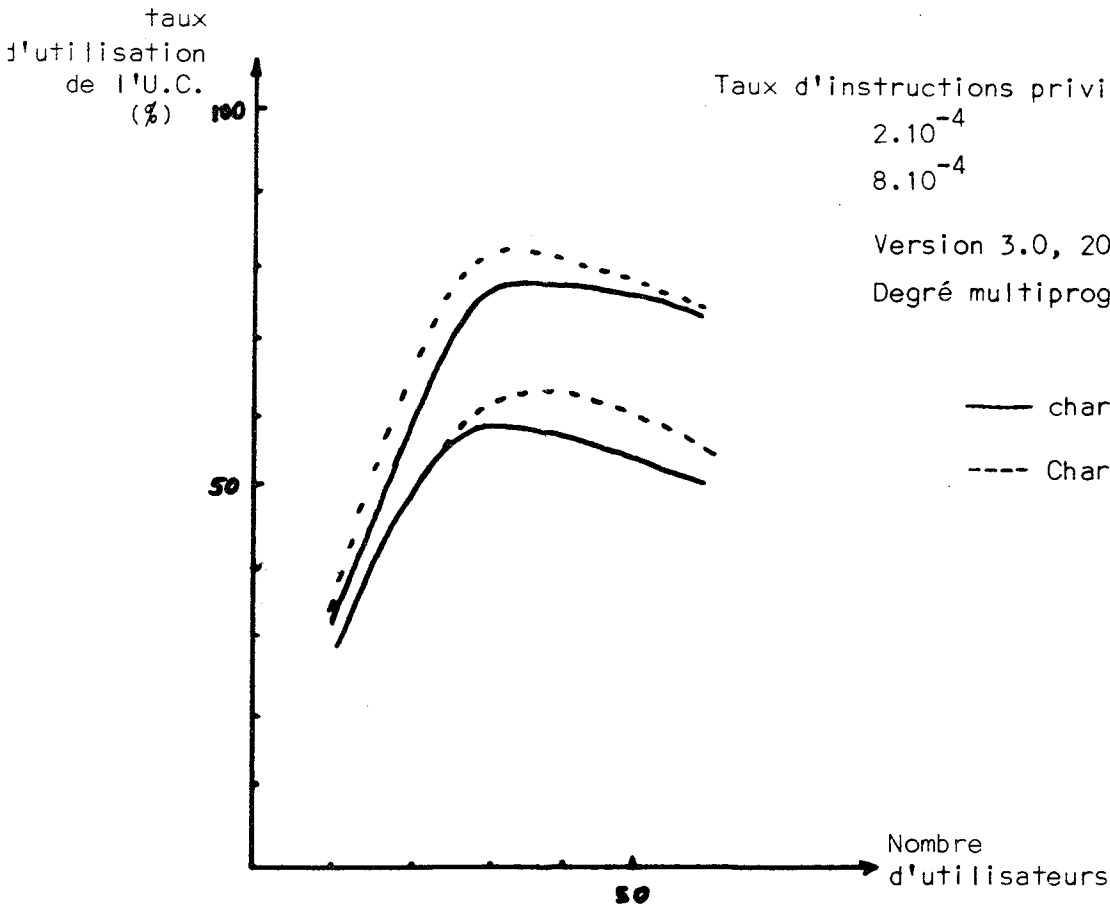


Figure 5.9.a.

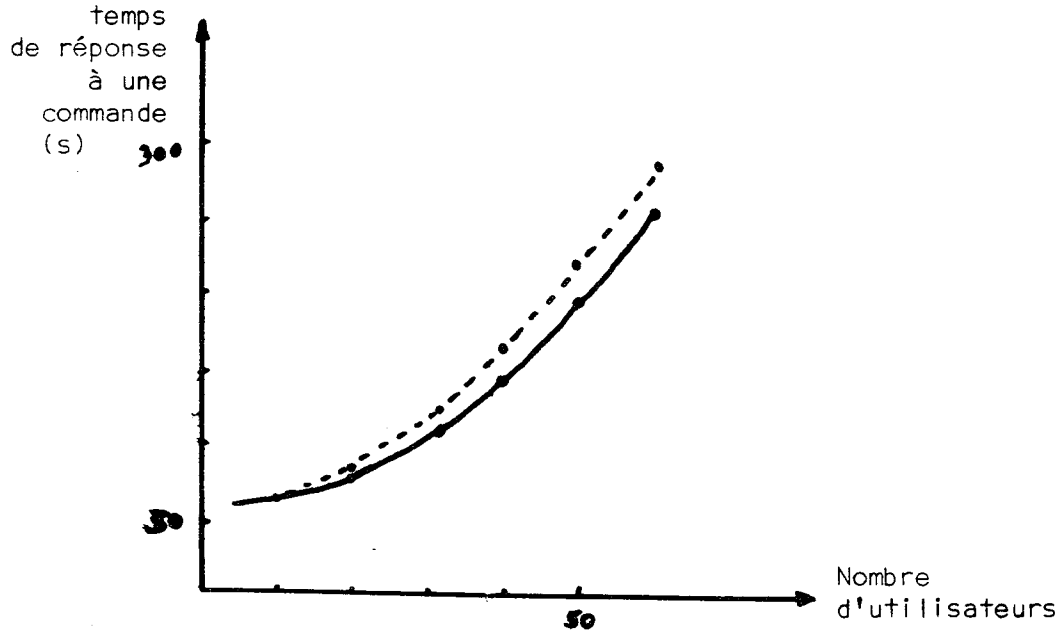


Figure 5.9.b.

Figures 5.9.a., 5.9.b. Influence du taux d'instructions privilégiées

On peut voir que le surcroît de travail n'est pas absorbé par l'attente car la proportion en mode problème a elle diminué légèrement (5 % environ). De même, les courbes de temps de réponse à la première machine virtuelle (5.9.b.) qui montrent un accroissement correspondant, confirme ce fait. Notons par ailleurs que ces courbes de temps de réponse ont l'allure caractéristique du modèle simple de "réparateur de machines" que Kleinrock [50] avait appliqué aux systèmes informatiques.

Si les tâches supplémentaires sont interruptibles ou abandonnables, ou bien déclenchées à des moments liés à leur durée, on peut certainement espérer un meilleur recouvrement. Dans le cas d'abandon néanmoins, un certain travail inutile aura été compté dans l'activité de l'Unité Centrale.

5.4.2. Influence de la répartition dans le temps des demandes d'entrées-sorties sur disque

Si les demandes d'entrées-sorties sur disque étaient uniformément réparties et indépendantes, le déplacement moyen du bras serait un tiers du nombre maximum de cylindres et non un demi. En effet, le bras se meut dans les deux sens et si X et Y sont deux variables aléatoires uniformes sur [0,1], le déplacement sera $|X-Y|$; or $E(|X-Y|) = \frac{1}{3}$

Pour un disque 2314, ce déplacement serait de 65 ms environ. Or nos mesures avec la charge C_2' indiquent un temps moyen de 15 ms. On retrouve donc un phénomène de localité dans les demandes successives d'entrées-sorties dû au fait que les fichiers d'un utilisateur sont sur des cylindres voisins en général et que les utilisateurs n'interfèrent pas trop (cf. Lynch [57]).

Remplaçons dans la charge C_2' les 30 entrées-sorties disque de l'initialisation par 2 entrées-sorties en initialisation et 4 dans chacune des 7 boucles du modèle de la commande, ce qui nous donne la charge C_3', nous avons refait les mesures de taux d'occupation des diverses ressources en fonction du nombre d'utilisateurs dans les mêmes conditions que l'expérience du § 5.3.3.

La figure 5.10 présente les résultats obtenus. Par rapport à la charge C_2', la proportion de temps problème a légèrement baissé, alors que le taux d'activité du canal a augmenté sensiblement : la durée moyenne d'une entrée-sortie sur disque est voisine de 20 ms. On a donc fait baisser la localité des demandes en les éparpillant dans le temps. La charge malgré tout assez faible du canal d'entrées-sorties disque ne justifie pas l'application de stratégies plus complexes que FIFO ([80]).

5.4.3. Importance des paramètres du système

Les expériences de Schatzoff et Tillman, relatées au § 2.1.3., faisaient varier la présence ou l'absence de cinq paramètres du contrôleur de travaux (version 3.2) afin de justifier ces "raffinements" pratiques. Ces cinq paramètres sont :

1. pénalité pour pagination excessive,
2. pénalité pour usage excessif de l'unité centrale,
3. maximum de pages pour une entrée-sortie en restant actif,
4. niveau maximum de multiprogrammation,
5. queue interactive à deux niveaux.

L'effet de ces paramètres a été étudié par 16 expériences qui combinent (non exhaustivement, mais suivant un plan d'expérience factoriel) leur absence et leur présence.

La proportion de temps en mode problème s'est révélée sensible au paramètre 3, au couple de paramètres (3,4), ainsi qu'au paramètre 4, alors que la proportion de temps en mode superviseur est sensible au cinquième paramètre. La méthode statistique utilisée est décrite dans le rapport [75].

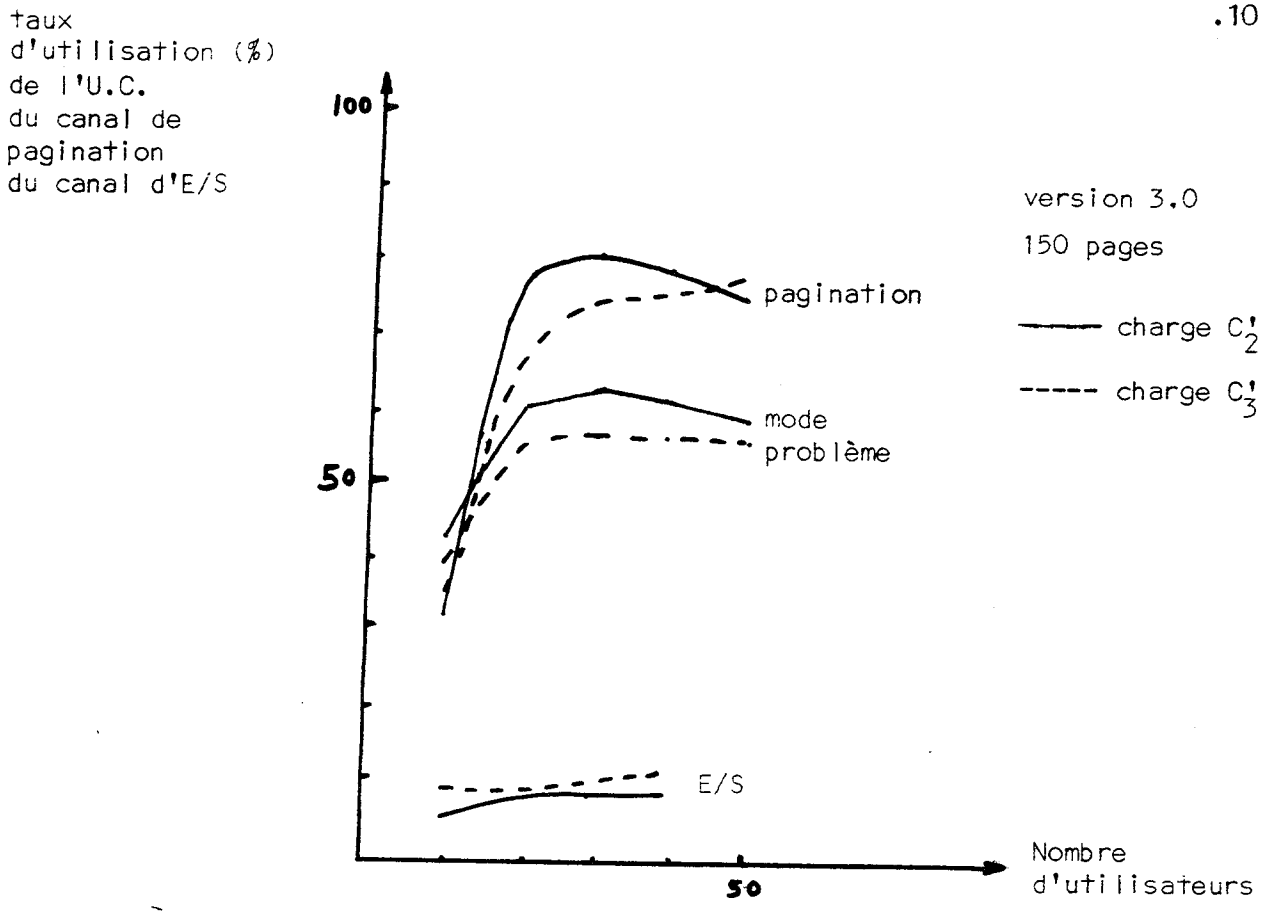


Figure 5.10. Influence de la répartition des demandes d'entrées-sorties dans le temps.

5.4.4. Autres usages du simulateur

Le simulateur a été utilisé accessoirement au Centre IBM de Cambridge : pour déterminer l'effet du partage de pages (Wahi [83]), pour étudier des méthodes de règlage du système ([76]), pour valider de nouvelles stratégies de contrôleur de travaux (Bard [8]), ainsi que pour mesurer les améliorations de performances apportées au système APL sous CMS ([21]).

Chapitre 6

C O N C L U S I O N S

Avant de livrer des réflexions issues de l'expérience acquise lors d'autres travaux ([15], [39], [13], [11], résumons les traits essentiels de ce travail et replaçons-le dans son contexte.

6.1. Le travail présenté et son contexte

J'espère avoir convaincu le lecteur que l'attente des ressources physiques dans un système informatique provient de deux sources :

. d'une part, du fait que les demandes concernent plusieurs ressources différentes qui ne seront donc pas toutes saturées en général, même en l'absence de contraintes de synchronisation ;

. d'autre part, du fait que la variabilité des demandes engendre, même sur les ressources qu'on pourrait croire saturées, une certaine attente.

J'espère aussi avoir montré comment la notion de processus, rapprochée de celle d'algorithme, permet d'exprimer séparément les parties d'un phénomène à étudier en vue de sa compréhension ou de sa simulation. En montrant que plusieurs structures de processus peuvent représenter le même phénomène (cf. Annexe 2), on invite l'utilisateur à choisir sa représentation plutôt que celle imposée par les systèmes de simulation.

La précision du principe de la simulation de processus et de ses limitations, la présentation des diverses fonctions des langages et systèmes de simulation, devraient permettre au lecteur de reconnaître les caractéristiques qu'il attend et ainsi de mieux choisir parmi les langages disponibles.

Le modèle de CP-67 illustre le fait que la structure de processus permet une bonne compréhension d'un grand système lorsque le niveau de fonctionnement à décrire est bien défini. Par exemple, ce modèle ignore les événements peu fréquents (connexion des machines virtuelles) ainsi que les particularités fines du système (structure du système d'interruptions), tout en décrivant l'essentiel de l'activité du système.

La phase de validation, souvent négligée, nécessite un effort de mesures et d'expériences assez grand avant qu'on puisse être raisonnablement certain de l'adéquation du modèle.

Les expériences de prédiction rapportées donnent des exemples de ce qu'il est possible d'attendre d'un tel modèle.

Signalons les thèmes suivants, non abordés dans ce travail, qui se rattachent à la simulation :

. Etude mathématique de la précision des réponses obtenues par simulation, soit en fonction du nombre d'échantillons observés ou du temps (cf. Le Gall [1], Geisler [34]), soit en fonction du domaine de l'environnement étudié (cf. Kempthorne [46], Davies [28]). Pour la précision dans le temps, il faut, au besoin par des mesures dynamiques, estimer l'influence des phénomènes transitoires et de l'auto-corrélation des variables. On peut aussi utiliser le fait que les suites pseudo-aléatoires sont reproductibles pour réduire la variance en introduisant des corrélations négatives entre variables aléatoires.

. Mise en forme ou applications de modèles mathématiques élaborés. Les modèles de réseaux de serveurs exponentiels ([65], [18], [6]) approchent le fonctionnement des systèmes réels sur des périodes assez longues (plusieurs semaines). Ils conviennent assez bien pour prédire l'effet de changements quantitatifs : tambours, disques, mémoires plus rapides, par exemple. Néanmoins, beaucoup de phénomènes n'apparaissent pas explicitement dans ces modèles : algorithmes de pagination, d'ordonnancement, par exemple.

Les recherches en cours concernent :

. la décomposition des systèmes en parties presque indépendantes (cf. Courtois [27], Mesarovic [63])

. l'étude des systèmes sous forme continue par les équations de "diffusion" (cf. Gelenbe [35]),

. l'étude de réseaux avec services arbitraires (cf. Chang et Lavenberg [24]).

Pour l'instant l'introduction de politiques de gestion, amorcée par Kleinrock [51] pour des systèmes simples, ne semble pas pouvoir se généraliser à des systèmes plus complexes.

A mon avis, si ces deux thèmes sont importants au point de vue mathématique, ils constituent un matériel sophistiqué qui risque de cacher la compréhension de phénomènes simples, comme l'inactivité des ressources par exemple.

. Comparaison systématique des langages de simulation, exemples détaillés de l'utilisation de certains langages (Schriber [78], Teichroew [79], Reitman [69], Emshoff [33]), rappel des expériences précédentes qui abordent l'essentiel des problèmes : modélisation, validation, prédiction (Nielsen [67], Scherr [77]).

Remarquons aussi que les changements technologiques redonnent un nouvel intérêt à des problèmes "classiques" d'évaluation : par exemple, quelles seront les performances d'un cache en présence d'un tambour de pagination ([11]), quel espace moyen devra-t-on attribuer à un programme en mémoire ? Si la pagination se faisait globalement, sans tenir compte de l'identité des tâches, par un mécanisme analogue à celui du cache, quelles performances atteindrait-on ?

Il semble donc qu'une activité de mesures et d'explication par des modèles simples doive toujours être associée à l'intervention de nouvelles solutions.

6.2. Les écueils de l'informatique

6.2.1. L'optimisation trop précoce

A mon avis, un des problèmes essentiels dans la conception et la réalisation des systèmes informatiques est l'introduction précoce ou à mauvais escient d'un souci "d'optimisation" qui conduit souvent à une complication inutile et même dangereuse.

Pour optimiser le fonctionnement d'un système, il faut de l'information sur les "demandes" qui vont lui être faites. Autrement dit, on n'optimise, ou l'on est "efficace", qu'en profitant des particularités de la demande. En général celles-ci sont trouvées à l'aide de mesures. Montrons-le à l'aide d'exemples.

Les algorithmes de hash-coding n'ont de propriétés intéressantes que lorsque la répartition des clés est supposée connue : Knuth ([52],vol 3 pp. 527-532) en fait l'analyse pour une répartition uniforme.

Dans CP-67, 95 % des demandes de blocs de mémoire (dite libre) ne concernaient qu'un petit nombre de tailles (5 à 10), principalement car elles provenaient d'éléments très répétitifs du système. On a donc modifié l'algorithme de gestion de ces blocs, basé sur la mise en listes par adresses croissantes et par tailles égales (cf. Ross [72]), en ne "fusionnant" pas les blocs de ces tailles fréquemment utilisées : le temps moyen par requête est ainsi passé de 300 µs à 45 µs (cf. Margolin [61]) !

Dans SIRIS 8, 95 % des demandes de réservation de blocs sur disques ne concernaient que quelques tailles (6), principalement car elles provenaient des "processeurs". En gardant un petit "stock" d'adresses de tels blocs en mémoire centrale, on n'utilise plus que 2 % des accès disque primitivement prévus, alors que la méthode standard favorisait les blocs d'un quantum, unité d'allocation qui vaut 8 k octets, en utilisant 30 % (cf. Machefaux [59]).

Dans CP-67 encore, en établissant un court-circuit pour la simulation des entrées-sorties disque de CMS, qui représentent plus de 90 % des entrées-sorties sur disque et qui sont de longueur fixe, on diminue l'overhead de CP de 30 à 50 % !

Dans OS 360, on fournit un éditeur de liens simplifié pour les travaux ne nécessitant aucun recouvrement ("overlay") ; un compilateur de Fortran, Watfor, compile rapidement (sans optimiser), engendre le code en mémoire, puis le fait exécuter.

En transcrivant une seule routine d'un programme de jeu d'échecs écrit en PL/1 en langage d'assembleur, on a divisé le temps de réponse par 3 (cf. Cassagne, Mailler [22]).

Donnons aussi quelques exemples des obstacles nés du souci trop précoce d'optimisation.

Fortran impose d'avoir des indices de tableaux de la forme a ± b :: i (Fortran II) ou que les valeurs initiales, finales et le pas d'une instruction DO soient obligatoirement une constante ou une variable. Plus généralement, le fait de ne pouvoir mettre n'importe quelle expression entière à la place d'un entier, complique la grammaire de ce langage, plutôt que de faciliter la compilation !

CP-67 remettait à zéro les pages en les recopiant d'une zone du tambour pour économiser les instructions qui auraient pu le faire. Malheureusement, des mesures (cf. Bard [7]) ont montré que la préparation d'une entrée-sortie sur tambour coûtait plus cher encore en unité centrale !

Le macro-assembleur du 360 interdit de redéfinir les instructions standard comme macro-instructions, ce qui empêche la transportabilité à partir des programmes IBM 360 par ré-assemblage et la transportabilité en sens inverse si des codes opérations coïncident !

De nombreuses limitations arbitraires existent dans les systèmes : pas plus de 19 morceaux de fichiers dans SIRIS 8, pas plus d'un disque virtuel en écriture sous CMS. Cela devient carrément "ubuesque" lorsque ces paramètres sont réglables : nombre maximum de bibliothèques utilisables simultanément dans CMS, nombre maximum d'éléments syntaxiques dans une expression d'assignation de PL/1, etc .. Qui peut les règler ? A quel prix ? Ce côté illusoire de la modularité, né du travail en équipe, sera évoqué plus tard.

Lorsque l'information relative aux demandes n'est pas disponible avant que le système existe, comment conduire la conception et la réalisation de ce dernier ? Il paraît logique de proposer la méthode suivante :

. Dans une première phase, réaliser aussi simplement (donc rapidement) que possible les fonctions nécessaires en incluant les moyens de mesure à la fois des caractéristiques de la demande et des performances de la réalisation.

. Dans une seconde phase, analyser les mesures afin de détecter les fonctions pour lesquelles on peut penser qu'il y a beaucoup à gagner (il faut en général qu'elles soient coûteuses) et se demander quelles particularités de la demande permettraient de le faire.

On diminuerait ainsi la complexité initiale des réalisations informatique et écarterait les "faux" problèmes, c'est-à-dire des problèmes qui se posent au niveau théorique mais jamais au niveau pratique. Par exemple, beaucoup de systèmes ne se protègent pas des "étreintes mortelles" (OS 360, SIRIS 8, CP 67 et Spartacus [15]) parce que la probabilité d'occurence est assez faible : en 7 ans d'utilisation de Spartacus, représentant plusieurs milliers d'heures de fonctionnement, un seul cas d'étreinte mortelle sur des buffers d'entrée-sortie s'est produit !

Sous CP, la demande de ressources se fait conversationnellement (disque supplémentaire) et par l'intermédiaire de l'opérateur (bandes). Le fait de répondre, même défavorablement à une demande de ressources permet à l'utilisateur de renoncer momentanément à son travail, donc aux ressources qu'il a pu accumuler pour sortir d'une éventuelle étreinte mortelle.

Cela ne veut pas dire bien sûr que les études théoriques sur la détection et la prévention des étreintes mortelles (cf. Haberman [38]) n'ont pas d'intérêt, mais qu'il ne faut les utiliser qu'à bon escient, plutôt que systématiquement.

De même, les politiques de gestion des entrées-sorties disques ont fait l'objet d'études théoriques et de simulations ([80]) ; mais sous une charge légère, qui se produit dans de nombreux systèmes ayant beaucoup d'utilisateurs potentiels, FIFO vaut toutes les politiques.

Malgré le souci de retarder les décisions apparentées à des optimisations, on ne peut le faire indéfiniment, tous les choix risquant d'être concernés : le tri entre les décisions immédiates et les autres relève de "l'art" de la programmation.

Notons néanmoins que certains obstacles surgissent pernicieusement : dans la méthode de "conception descendante" d'un système, où les fonctions complexes sont décomposées en algorithmes utilisant des fonctions plus élémentaires, le langage qui permet l'expression de ces algorithmes nous force à faire des choix sur les structures données, car les notations d'une variable, d'un élément de tableau, d'un élément de liste ou du résultat d'une procédure sont généralement distinctes ; écrire ces algorithmes entraîne donc le choix arbitraire de la structure des données associées.

6.2.2. Autres sources de complications

La notion de "compliqué" que nous nous efforçons d'éliminer est subjective et, au fond, n'est compliqué pour nous que ce que nous ne comprenons pas bien.

Ainsi les programmes ou systèmes qui nous paraissent compliqués peuvent l'être soit parce que leurs auteurs n'ont pas compris eux-mêmes ce qu'ils ont fait, soit parce que l'explication de leurs travaux n'est pas à la hauteur de ceux-ci.

Le premier cas est plus fréquent que l'on ne pense :

. maladresse de l'exécutant qui réalise des programmes confus pour un travail facile et clair a priori,

. ignorance de techniques éprouvées ou de concepts déjà formulés, qu'un travail (ou une recherche) va redécouvrir partiellement et progressivement,

. mauvaise coordination dans le travail de plusieurs personnes qui n'utilisent pas les mêmes techniques, "optimisent" séparément et de façon contradictoire ou n'ont pas la même idée de la simplicité.

Les langages de commande entrent dans ce dernier cas : imaginez un voyageur qui ne comprend pas la langue du pays visité (langage machine), mais qui a appris une langue pour laquelle il sait que des interprètes existent (Algol). Eh bien, il lui faut encore un autre langage ("JCL") pour demander cet interprète. Le fait d'avoir des langages de commande proches du langage de travail faciliterait la tâche de l'utilisateur. Cela demande peut-être d'avoir plusieurs versions des fonctions de commande suivant que l'utilisateur emploie Cobol, Fortran, Algol ou Lisp.

C'est dans l'effort de réflexion pour exprimer de façon satisfaisante l'explication des travaux qui ont été réalisés, que d'une part la technique rejoint la recherche et que l'on arrive à transmettre une partie de son "expérience", d'autre part.

Au cours d'une telle réflexion émergent des concepts nouveaux, se fait la synthèse de précédents concepts et se trouvent relégués au rang de détails certains aspects qui risquent de nuire à la compréhension.

Par exemple, c'est l'équivalence entre l'analyse d'une grammaire formelle par un automate ou un programme qui peut conduire à incorporer à celle-ci des "macro-instructions" (méta-règles) ou des "variables" (dépendance entre règles et contextes) [13].

C'est en reconnaissant que les transformations sur des figures peuvent être arbitraires plutôt que liées aux transformations dans l'espace à deux ou trois dimensions, qu'on voudrait les exprimer par un algorithme dans un langage comme Algol ou Fortran et qu'on greffe ainsi un langage graphique sur un langage algorithmique [39].

C'est en voyant que par microprogrammation on utilise une machine étendue par un interpréteur (les micro-programmes) que l'on peut banaliser le "cache" afin qu'il accueille microprogrammes, programmes, données suivant la même règle d'usage récent.

Entre la communication de messages par interruption et par boîte aux lettres, on peut envisager de mettre ces messages dans une boîte, munis d'une minuterie ("time-out"). Ainsi le temps de prise en compte est garanti, sans que le travail en cours soit systématiquement interrompu.

Le genre d'aspects qu'il faut plutôt cacher peut être illustré ainsi le PDP 8, support du système Spartacus, était conçu comme un concentrateur de terminaux. Pour gérer trente terminaux, il passe environ la moitié de son temps à échantillonner huit fois tous les bits de chaque caractère envoyés en série !

En parlant de ce système, il me paraissait important d'expliquer complètement ce mécanisme, dont la programmation fait, en volume, un bon tiers du superviseur ! Aujourd'hui, je pense que cela ne faisait qu'embrouiller la simplicité du principe de partage de temps par "swapping" qui permettait à Spartacus d'être conversationnel.

6.3. Au-delà de l'informatique

La modélisation et la simulation en informatique semblent pouvoir intervenir à trois niveaux, vis à vis des autres disciplines scientifique:

Premièrement, en tant que technique, l'informatique est de plus en plus employée pour l'analyse de données s'appuyant sur des théories mathématiques : statistiques (corrélation, analyse de la variance) ou "optimisation" (segmentation, nuées dynamiques). Un danger évident est que le spécialiste d'une autre discipline utilise aveuglément ces moyens qui peuvent coûter fort cher et qui donneront toujours des résultats numériques. Compte tenu de la sophistication de ces méthodes, il semble souhaitable qu'un dialogue permanent s'installe entre les utilisateurs et les mathématiciens pour guider leur bon usage.

Toujours en tant que technique, l'informatique interviendra lorsque les disciplines voudront élaborer des modèles dont le comportement ne peut être étudié que par simulation. Les modèles cybernétiques comportant des servo-mécanismes plus ou moins complexes, sont envisagés par de nombreuses disciplines :

. en biologie, aussi bien au niveau du métabolisme d'une cellule [64] qu'au niveau de l'individu [53] en renforçant la distinction entre information et énergie ;

. en économie, où les interactions entre régions du monde [63] , l'inclusion du monde économique dans l'éco-sphère et l'introduction de la notion d'information [3], bouleversent les notions limitées à la production et à la consommation et leurs modèles mathématiques.

Non seulement ces disciplines pourront trouver l'outil adéquat en informatique, mais les informaticiens ou les mathématiciens se trouvant au carrefour des essais des divers modèles pourraient participer à leur invention.

Deuxièmement, la méthodologie d'utilisation des modèles de simulation est complexe et pose de nombreux problèmes théoriques : précision, validation, passage d'un niveau de "finesse" du modèle à un autre. L'informatique peut alors être utilisée comme objet de simulation, à l'aide duquel on pourrait progresser dans les problèmes précédents.

En effet, le monde réel des systèmes informatiques (les programmes et les matériels) peut être décrit avec une assez grande finesse et les expériences sont assez bien reproductibles. La simulation des systèmes informatiques, outils assez coûteux, en vue de leur optimisation, présente un intérêt économique évident.

Troisièmement, les problèmes spécifiques de l'informatique pourraient bien être la clé d'autres connaissances. En informatique, la notion de "boîte noire" s'étend partout. Une instruction peut être câblée, microprogrammée, simulée par programme après interruption (instruction optionnelle ou instruction privilégiée d'une machine virtuelle) ; ce qui compte c'est son résultat et non la manière dont il a été obtenu. De même, les notions de sous-programme, procédure, module, obéissent encore à cette règle. La notion d'interpréteur à un niveau supérieur, simule une machine langage : c'est aussi une "boîte noire". On peut donc caractériser l'informatique comme la technique d'organisation de ces boîtes noires pour en fabriquer de plus grandes.

On conçoit dès lors qu'il peut y avoir des rapports entre l'activité cérébrale de l'homme et l'informatique. Les techniques que cette dernière emploie pourraient suggérer des modèles pour la mémorisation, la recherche de l'information ou l'apprentissage humain. Inversement, les fonctions humaines expliquées pourraient se transposer pour l'ordinate

On risque ainsi de démystifier certains concepts comme l'intelligence, d'une part en montrant qu'elle recouvre des notions très différente et, d'autre part, en rendant des machines capables de tâches réputées intelligentes. Par exemple, jouer aux échecs est une de ces tâches.
Le progrès dans ce domaine pourrait bien venir de l'expression de structure de données capables de représenter les relations qu'un joueur voit entre les pièces sur l'échiquier. Or, nous sommes capables de jouer, mais pas d'exprimer logiquement pourquoi et à l'aide de quels renseignements (l'enseignement aux échecs se fait d'ailleurs presque uniquement par l'exemple).

Enfin, la place de l'informatique dans la société est en gestation. Avec la prééminence de l'information sur l'énergie, les applications informatiques risquent de jouer un rôle analogue à celui de l'électricité d'aujourd'hui. De nombreuses décisions pouvant être automatisées le seront, ce qui entraînera l'enregistrement de l'information qui leur est nécessaire.

Les informaticiens, surtout au niveau de la recherche et de l'enseignement, devront assumer la double responsabilité, d'une part de diffuser l'information sur l'objet de leurs recherches et sur les dangers éventuels qui pourraient en résulter et, d'autre part, de susciter l'élaboration d'une éthique de l'informatique par la société ou l'humanité, puis de l'appliquer.

B I B L I O G R A P H I E

[1] AGARD J,
Les méthodes de simulation
Dunod, 1968

[2] ANCEAU F,
Contribution à l'étude des systèmes hiérarchisés de ressources dans l'architecture des machines informatiques
Thèse de Doctorat d'Etat, Grenoble, 1974

[3] ATTALI J,
La parole et l'outil
PUF, 1975

[4] AUROUX A & HANS C,
Le concept de machines virtuelles
RAIRO n° 15, 1968

[5] BACHELARD G,
Essai sur la connaissance approchée
Vrim, 1973

[6] BAIXAS D,
Exemple d'utilisation du modèle de Gordon et Newell
Journées Mesures et Modèles, Toulouse, Juin 1973

[7] BARD Y,
CP 67 measurement and analysis : overhead and throughput
ACM Sigops Workshop on System Performance Evaluation, Avril 1971

[8] BARD Y,
Application to page survival index (PSI) to virtual memory system performance
IBM Cambridge Scientific Center Report, Juin 1974

[9] BELADY L.A. & KUEHNER C.J.,
Dynamic space-sharing in computer systems
C.ACM, Mai 1969

[10] BELADY L.A., NELSON R.A. & SHEDLER G.S.,
An anomaly in space-time characteristics of certain programs running in a paging machine
C.ACM, Juin 1969

[11] BENNETT M, BERARD P, BOKSENBAUM C, VERAN M,
Notes on the efficiency of a cache memory
Informatica 74, Bled, Octobre 1974

[12] BETOURNE C. & KRAKOWIAK S,
Simulation d'un système conversationnel à mémoire virtuelle
Journées Mesures et Modèles, Toulouse, Juin 1973

[13] BOKSENBAUM C,
Moyens et problèmes de la compilation sur petites machines
RIRO 1970

[14] BOKSENBAUM C, GREENBERG S, TILLMAN C,
Simulation of CP 67
IBM Cambridge Scientific Center, Juin 1973

[15] BOKSENBAUM C, GUIBOUD RIBAUD S,
Etude et réalisation d'un système pédagogique de programmation en ligne et en temps partagé
Congrès AFIRO, Lille, Mai 1967

[16] BORRIONE D,
Une analyse de système par typologie
Colloque IRIA sur les Systèmes d'Exploitation, Avril 1974

[17] BOURBAKI N,
Eléments d'histoire des mathématiques
Hermann, 1960

[18] BRANDWAJN A, GELENBE E, LENFANT J, POTIER D,
Modèle de système multiprogrammé à mémoire virtuelle paginée
Journées Mesures et Modèles, Toulouse, Juin 1973

[19] BUZEN J.P.,
Analysis of system bottlenecks using a queuing network model
ACM Sigops Workshop on System Performance Evaluation, Avril 1971

[20] BUZEN J.P.,
Computational algorithms for closed queuing network with exponential servers
C.ACM, Septembre 1973

[21] BUCO W.M, CRISTOFORS A.J, HATFIELD J, TILLMAN C.C,
A methodology for paging performance enhancements
Manuscrit 1973

[22] CASSAGNE B, MAILLER A,
Programme d'aide au jeu d'échecs
Rapport de Projet, ENSIMAG, Juin 1972

[23] CHAMBERLIN D.D, FULLER S.H, LIU L.Y,
A page allocation strategy for multiprogramming systems with virtual memory
Fourth Symposium on Operating System Principles, 1973

[24] CHANG, LAVENGERG
Work-rates in closed queuing networks with general independent servers
San Jose IBM Research Laboratory Report, 1971

[25] CODD E.F,
A relational model of data for large shared data banks
C.ACM, Juin 1970

[26] COFFMAN E.G,
Analysis of a drum I/O queue under scheduler operations in a paged computer system
J.ACM, Janvier 1969

[27] COURTOIS P.J,
On the near-decomposability of networks of queues and of stochast models of multiprogramming computing systems
Rapport MBLE, 1971

[28] DAVIES O,
Design and analysis of industrial experiments
Hefner, 1960

[29] DENNING P.J,
The working-set model of program behaviour
C.ACM, Mai 1968

[30] DENNING P.J,
Thrashing : its cause and prevention
AFIPS SJCC, 1968

[31] DIJKSTRA E.W,
Solution of a problem in concurrent programming control
C.ACM, Septembre 1965

[32] DIJKSTRA E.W,
The structure of THE multiprogramming system
C.ACM, Mai 1968

[33] EMSHOFF J.R, SISSON R.L,
Design and use of computer simulation models
Mac Millan, 1970

[34] GEISLER M.A,
The size of simulation samples required to compute certain inventory characteristics with stated precision and confidence
Management Science n° 6, 1964

[35] GELENBE E.,
On approximate computer system models
Computer Architecture and Networks Workshop, IRIA, 1974

[36] GOLDBERG R.P,
Architecture of virtual machines
AFIPS NCC, 1973

[37] GORDON W.J, NEWELL G.F,
Closed queuing systems with exponential servers
Operations Research, n° 2, 1967

[38] HABERMAN A.N,
Prevention of system deadlocks
C.ACM, Juillet 1969

[39] HAOUR J.L, BREHINIER S, BOKSENBAUM C,
LAPREC : langage de programmation pour écran cathodique
RIRO n° 15, 1968

[40] HATFIELD D.J,
Experiments on page size program access patterns and virtual memory performance
IBM Journal Research & Development, 16.1, Janvier 1972

[41] HILL I.D,
Wouldn't be nice if we write computer programs in ordinary English, or would it ?
The Computer Bulletin, Juin 1972

[42] HORNING J.J. & RANDELL B,
Process structuring
ACM Computing Surveys, Mars 1973

[43] JACKSON R.R.P,
Jobshop-like queuing systems
Management Science, 10, 1963

[44] JONES D,
Physics today 1970
Résumé dans Scientific American, mai 1970, p. 58, repris par Science et Vie, Février 1971

[45] JONES M.M,
Incremental simulation on a time-shared computer
PhD. Thesis, MAC, TR-48, MIT, Janvier 1968

[46] KEMPTHORNE O,
The design and analysis of experiments
Wiley, 1952

[47] KIVIAT P.J, VILLANUEVA R,
The SIMSCRIPT II Programming Language
Prentice Hall, 1969

[48] KIVIAT P.J., COLKER A,
Gasp : a general activity simulation program
Rapport de la Rand Corp., 1964

[49] KLEINROCK L,
Resource allocation in computer systems and computer communication networks
IFIPS Proceedings, 1974

[50] KLEINROCK L,
Certain analytical results for time-shared processors
IFIPS Proceedings, 1968

[51] KLEINROCK L,
A continuum of time-sharing scheduling algorithms
AFIPS SJCC, 1970

[52] KNUTH D.E,
The art of computer programming
Addison Wesley, 1973

[53] LABORIT H,
La nouvelle grille
R. Laffont, 1974

[54] LAMPORT L,
A new solution of Dijkstra's concurrent programming problem
C.ACM, Août 1974

[55] LEROUDIER J,
Une analyse de système
Thèse de Troisième Cycle, Grenoble, 1973

[56] LIPTAY J.S,
Structural aspects of the system 360 model 85 - the cache
IBM Systems Journal, vol 7 n° 1, 1968

[57] LYNCH W.C,
Do disk arms move ?
ACM SIGME Performance Evaluation Review, Décembre 1972

[58] MAC DOUGALL M.H,
Computer system simulation. An introduction
ACM Computing Surveys, Septembre 1970

[59] MACHEFAUX J.P,
Etude et amélioration d'un algorithme d'allocation d'espace sur disques
Thèse de Troisième Cycle, Grenoble, 1974

[60] MAHL R,
An analytical approach to computer systems scheduling
PhD. Thesis, University of Utah, Juin 1970

[61] MARGOLIN B.H, PARMELEE R.P, SCHATZOFF M,
Analysis of free-storage algorithms
Statistical Computer Performance Evaluation (Freiberger ed.)
Academic Press, 1972

[62] MARKOV A,
Theory of algorithms
Press for Scientific Translations, 1961

[63] MESAROVIC M, PESTEL E,
Stratégie pour demain
Seuil 1974

[64] MONOD J,
Le hasard et la nécessité
Le Seuil, 1973

[65] MOORE C.G,
Network models for large-scale time-sharing systems
PhD. Thesis, University of Michigan, Avril 1971

[66] MORGANSTEIN, WINOGRAD, HERMAN,
SIM/61 : a simulation measurement tool for a time-shared demand paging operating system
ACM Sigops Workshop on System Performance Evaluation, Avril 1971

[67] NIELSEN N.R,
The simulation of time-sharing systems
C.ACM, Juillet 1967

[68] PUGH A.L,
Dynamo user's manual
MIT Press, 1961

[69] REITMAN J,
Computer Simulation applications
Wiley, 1971

[70] RODRIGUEZ ROSELL J,
The evaluation of a time-sharing page demand system
AFIPS SJCC, 1972

[71] RODRIGUEZ ROSELL J, DUPUY J.P,
The design, implementation and evaluation of a working-set dispatcher
C.ACM, Avril 1973

[72] ROSS D.T,
Introduction to software engineering with the AED language
MIT Report, ESL-405

[73] ROUSSET DE PINA X,
Modèle déterministe pour l'étude de l'efficacité d'un système en temps partagé
RIRO B.3, 1971

[74] SALTZER J.M,
A simple linear model of demand paging performance
C.ACM, Avril 1974

[75] SCHATZOFF M, TILLMAN C,
Design of experiments in simulator validation
IBM Cambridge Scientific Center, Juin 1974

[75 bis] SCHATZOFF M, WHEELER L.H.,
CP/67 Paging priority dispatcher
IBM Cambridge Scientific Center Report, Mars 1973

[76] SCHATZOFF M, BRYANT P,
Regression methods in performance evaluation : some comments on the state of the art
Proceedings of Computer Science and Statistics, Seventh Annual Symposium, Iowa State University, Octobre 1973

[77] SCHERR A.L,
An analysis of time-shared computer systems
MIT Press, 1967

[78] SCHRIBER T.J,
Simulation using GPSS
John Wiley & Sons, 1974

[79] TEICHROEW D, LUBIN J.F,
Computer simulation. Discussion of the technique and comparison of languages
C.ACM, Octobre 1966

[80] TEOREY T.J, PINKERTON T.B,
A comparative analysis of disk scheduling policies
C.ACM, Mars 1973

[81] VAN HORN E.C,
Three criteria for designing computing systems to facilitate debugging
C.ACM, Mai 1968

[82] VAUCHER J.G,
La programmation avec SIMULA 67
Université de Montreal, Juillet 1973

[83] WAHI P.J,
On sharing of pages in CP 67
ACM Workshop on Virtual Computing Systems, Harvard, Mars 1973

[84] WIRTH N,
The programming language Pascal
Acta Informatica, vol 1 fasc. 1, 1971

APPENDICE 1

Croissance de l'activité dans un réseau fermé de serveurs exponentiels

Considérons un ensemble de m serveurs tels que la durée de service du serveur $i (1 \leq i \leq m)$ soit une variable aléatoire suivant une loi exponentielle de moyenne $\frac{1}{a_i}$ et tel que le client sortant de ce serveur soit dirigé vers le serveur k avec la probabilité p_{ik}. (On a donc $\forall\ i \in [1,m]$, $\sum_{j=1}^{m} p_{ij} = 1$).

La distribution des clients dans un tel système a été étudié par Jackson [43], Gordon et Newell [37], Moore [65] et Buzen [19], [20], dont nous utiliserons les notations.

A l'état stationnaire, lorsque la matrice P d'élément P_{ij} est irréductible, la probabilité d'avoir $(n_1,\ldots, n_m)$ clients, en service ou en attente, aux serveurs respectifs 1,...m, est :

$$p(n_1,\ldots n_m) = \frac{1}{g(n,m)} \prod_{i=1}^{m} x_i^{n_i} \qquad (1)$$

où $g(n,m) = \sum_{\Sigma n_i = n} \prod_{i=1}^{m} x_i^{n_i}$ somme des monômes de degré n en $x_1 \ldots x_m$, est un facteur de normalisation.*

où les x_i sont tels que le vecteur ligne $Y = (a_i x_i)_{i \in [1,m]}$ est la solution réelle positive de l'équation YP = Y, définie à une constante multiplicative près.
La probabilité d'avoir au moins k clients au serveur j est :

$$p(n_j \geq k) = x_j^k \frac{g(n-k,m)}{g(n,m)}$$

En particulier la probabilité d'avoir au moins un client, qui est aussi la proportion de temps où l'unité est active, sera :

$$p(n_j \geq 1) = x_j \frac{g(n-1,m)}{g(n,m)}$$

Nous voulons montrer que cette quantité croît strictement avec n, c'est-à-dire que :

$$\frac{g(n-1,m)}{g(n,m)} < \frac{g(n,m)}{g(n+1,m)} \qquad \forall\ x_j > 0 \quad j=1,\ldots m \qquad \forall\ n > 0\ , \forall\ m > 1$$

où $g(n,m)^2 > g(n-1,m)\ g(n+1,m)$

* qu'on appelle polynome de Perron et que l'on note plus concisément : $\sum_{|a|=n} Y^a$

Auparavant démontrons 3 lemmes :

Lemme 1 : $g(n,m) = \sum_{k=0}^{m} x_m^k \, g(n-k,\ m-1) \qquad \forall n > 0, \ \forall m > 1$

dem En séparant les monômes de $g(n,m)$ qui contiennent x_m des autres, on obtient :

$$g(n,m) = g(n,\ m-1) + x_m g(n-1,m-1)$$

comme $g(0,m) = 1, \forall\, m \geq 0$, on obtient par récurrence :

$$g(n,m) = \sum_{k=0}^{n-1} x_m^k \ g(n-k,\ m-1) + x_m^n \, g(0,m) = \sum_{k=0}^{n} x_m^k \, g(n-k,m-1)$$

Lemme 2 $\quad g(n,2)^2 - g(n-1,2)\ g(n+1,2) > 0 \ \forall n > 0$

appelons D le premier membre

dem. 1er cas : $x_1 \neq x_2$ $(x_1,\ x_2 > 0)$.

comme $(x_1 - x_2)\ g(n,2)^2 = x_1^{n+1} - x_2^{n+1} \quad \forall\ n \geq 0$,

$(x_1 - x_2)^2$ D, de même signe que D, est égal à :

$$(x_1^{n+1} - x_2^{n+1})^2 - (x_1^n - x_2^n)(x_1^{n+2} - x_2^{n+2}) =$$

$$(x_1x_2)^n \quad [-2x_1x_2 + x_1 + x_2] \ = (x_1x_2)^n(x_1 - x_2)^2$$

donc $D = (x_1x_2)^n > 0$

2ème cas : $x_1 = x_2$ $(x_1 > 0)$

$g(n,m)$ possédant C_{n+m-1}^n monômes, on a :

$$D = x_1^n \ [(n+1)^2 - n(n+2)] \ = x_1^n > 0$$

Lemme 3 : si l'on a, pour m fixé

$$g(n,m)^2 > \ g(n-1,m)g(n+1,m) \quad \forall\, n > 0$$

cela entraîne que

$$g(n-k,m)\ g(n,m) > g(n-k-1,m)\ g(n+1,m) \quad \forall\ k < n$$

dem la première relation s'écrit :

$$\frac{g(h,m)}{g(h-1,m)} < \frac{g(h+1,m)}{g(h,m)} \qquad \forall\ h > 0$$

en faisant le produit membre à membre des relations pour h allant de n-k à n, on obtient

$$\frac{g(n-k,m)}{g(n-k-1,m)} < \frac{g(n+1,m)}{g(n,m)} \qquad \forall\, n > 0$$

c'est-à-dire la relation cherchée.

Nous pouvons démontrer par récurrence la propriété :

Proposition : $\forall\, n > 0,\ \forall\, m > 2,\ g(n,m)^2 - g(n-1,m)\ g(n+1,m) > 0$

dem . nous allons démontrer cette relation par récurrence sur m. Comme elle est vraie pour m = 2 (lemme 2) , il nous reste à démontrer que si la relation est vraie pour m-1 cela entraîne qu'elle est vraie pour m.

En utilisant le Lemme 1 et en posant $g(x,m-1) = h(x)$ la relation cherchée s'écrit ;

$$\sum_{h=0}^{2n} x_m^i \left(\sum_{\substack{k_1+k_2=i \\ 0 \le k_1 \le n \\ 0 \le k_2 \le n}} h(n-k_1)h(n-k_2) - \sum_{\substack{k_1+k_2=i \\ 0 \le k_1 \le n-1 \\ 0 \le k_2 \le n+1}} h(n-1-k_1)h(n+1-k_2) \right) > 0$$

Montrons que tous les coefficients de ce polynôme en x_m sont positifs et nuls, certains étant strictement positifs.

Dans la première somme k_2 varie de i-min(i,n) à min(i,n) donc $j=n-k_2$ varie de n-min(i,n) à n-i+ min(i,n). Posons s = 2n-i et $j' = n-k_2$. Le coefficient de x_m^i s'écrit donc :

$$\sum_{j=n-\min(i,n)}^{n-i+\min(i,n)} h(s-j)h(j) - \sum_{j'=n-1-\min(i,n-1)}^{n-1-i+\min(i,n+1)} h(s-j')h(j')$$

Nous devrons donc comparer pour les diverses valeurs de i les intervalles ($\subset N$) :

$[n-\min(i,n),\ n-i+\min(i,n)]$ et $[n-1-\min(i,n-1),\ n-1-i+\min(i,n+1)]$

1er cas : $0 \le i \le n-1$

les 2 intervalles sont $[n-i,\ n]$ et $[n-1-i,\ n-1]$

Le coefficient se réduit à

$$h(s-n)\ h(n) - h(s-n+1+i)h(n-1-i) = h(n-i)h(n) - h(n+1)h(n-i-1)$$

or, si la propriété est vraie pour m-1, le Lemme 3 , indique que l'expression ci-dessus est positive, $\forall\ i \le n-1$.

2ème cas : i=n

Les intervalles sont alors

$$[0,\ n\ ] \quad \text{et} [0,\ n-1]$$

Le coefficient est alors $h^2(n) > 0$

3ème cas : $i \geq n+1$

Les intervalles sont $[0,\ 2n-i\ ]$ et $[0,\ 2n-i\ ]$: le coefficient est nul.

Comme le second cas existe toujours, la différence est bien strictement positive, et la proposition est démontrée.

Cette démonstration, plus courte que la démonstration initiale nous a été suggérée par M. MUNTEAN. La relation plus générale

$$g(a,m)\ g(b,m) > g(a',m)\ g(b',m)$$

si $a+b = a'+b'$ et $\max(a',b') > \max(a,b)$

peut facilement s'en déduire.

En effet, en supposant $a > b$ et $a' > b'$ ce qui n'enlève rien à la généralité,

$$\frac{g(a,n)}{g(a-1,m)} < \frac{g(a+1,m)}{g(a,m)} \rightarrow \frac{g(a,m)}{g(a-1,m)} < \frac{g(a',m)}{g(a'-1,m)} \qquad \forall\ a \geq 0,\ \forall\ a' > a$$

en multipliant membre à membre les inégalités précédentes prises de $b'+1$ à a on obtient :

$$\frac{g(a,m)}{g(b',m)} < \frac{g(a',m)}{g(a'+b'-a)} = \frac{g(a',m)}{g(b,m)}$$

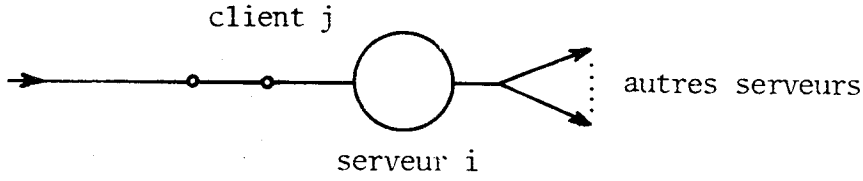
C Q F D

Appendice 2

Exemple de dualité dans l'expression d'algorithmes concurrents

Nous voulons représenter un ensemble de n clients qui transitent entre m serveurs de la façon suivante :

chaque client se présente à un poste de service où, après éventuellement avoir attendu, il sera servi et repartira vers un autre poste de service.



Les actions que nous voulons décrire sont les services que le serveur j rend au client i et nous supposerons que la procédure service (i,j) est capable de faire ces descriptions.

Nous pouvons décrire ces actions en attachant des processus soit aux serveurs, soit aux clients.

1. Processus utilisateurs

n processus ayant la structure suivante (en pseudo Algol 68) :

```
procédure utilisateur = (entier j) neutre :
   début i suivant := i initial(j) ;
      faire début i := i suivant ;
                  P(s(i)) ;
                  service(i,j) ;
                  i suivant := serveur suivant(j) ;
                  v(s(ii)) ;
      fin fait
   fin
```

représentent(pour j allant de 1 à n) l'activité de l'utilisateur j qui fait appel au serveur i, comme à une ressource par l'intermédiaire du sémaphore s(i)

Lorsqu'il l'obtient, il effectue le service, détermine le prochain serveur dont il aura besoin avant de libérer le serveur qu'il quitte.

Ces n processus lancés en parallèle décrivent le phénomène que nous voulions étudier.

2. Processus serveurs

m processus ayant la structure suivante :

```
procédure serveur = (entier i) neutre :
  début nombre initial (i) faire v(s'(i)) ;
    faire début
            P(s'(i)) ;
              p(mutex(i)) ;
              j := sélectionné(i) ;
              v(mutex(i)) ;
                service(i,j) ;
                i suivant := serveur suivant(j) ;
              p(mutex(i suivant)) ;
              insérer(i suivant,j) ;
            v(s'(i suivant)) ;
              v(mutex(i suivant)) ;
    fin fait
  fin
```

représentent (pour i allant de 1 à m) l'activité du serveur i qui se demande s'il a des clients par l'intermédiaire d'un sémaphore qui lui est propre s'(i). S'il en a, il sélectionne un utilisateur, le sert, détermine le serveur suivant que cet utilisateur demandera, puis l'insère dans l'ensemble des utilisateurs attendant ce serveur (d'indice i suivant) avant de prévenir le serveur qu'il a un client supplémentaire.

On remarquera que les actions de sélection et d'insertion sont ici explicites, alors que les sémaphores attachés aux serveurs ne servent que de compteurs des clients en attente. Ces actions doivent être faites de façon exclusive : un sémaphore mutex (de mutuelle exclusion) attaché à chaque serveur est utilisé à cette fin.

Dans la première description p(s(i)) rend le client j candidat pour le serveur j, tandis que dans la seconde v(s'(isuivant)) rend le client j candidat pour le serveur isuivant.

De même, dans la première description v(s(i)) indique que le serveur i libébré par le client j, peut servir un autre client s'il y en a, tandis que p(s'(i)) indique que le serveur i peut servir un autre client dans la seconde description.

Ainsi dans ces représentations duales, les actions p et v sont aussi duales.

Pour choisir parmi ces représentations, on peut tenir compte des arguments suivants :

- la représentation des processus est plus coûteuse que celles de tâches en général; On peut choisir les entités (clients ou serveurs) les moins nombreuses comme processus.
- si une seule des entités fait des actions significatives sans être associée à l'autre, on peut la représenter par des processus, où ces actions seront décrites.

Lorsque les deux entités peuvent agir séparément puis conjointement, il paraît nécessaire de n'écrire l'action commune que d'un seul point de vue.

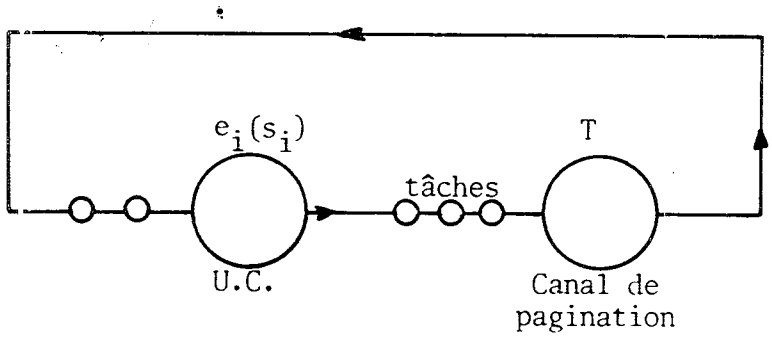
Appendice 3

ETUDE DU TAUX D'ACTIVITE D'UN SYSTEME

AVEC PAGINATION AVEC TEMPS DE SERVICE CONSTANT

°

Considérons un système analogue à celui du § 2.2. où la tâche i, parmi n, provoque un défaut de page tous les $e_i(s_i)$ lorsque s_i pages lui sont allouées en mémoire centrale. T est le temps, supposé constant, de transfert d'une page.



a. Etude de l'écroulement du système

Nous supposons que toutes les tâches sont semblables, c'est-à-dire ont la même fonction "durée de vie" e(s) croissante avec s et le même espace en mémoire centrale $\frac{M}{n}$.

Appelons s_T la valeur de s telle que $e(s_T) = T$ et $n_T \triangleq \frac{M}{s_T}$.

Nous supposons qu'à partir d'une valeur $s_0 > s_T$, qui représente l'espace maximum que nécessite une tâche pour être exécutée, la fonction e(s) reste constante :

$$\forall s \geq s_0 \qquad e(s) = e(s_0)$$

La condition exprimant que l'unité centrale n'est pas saturée se réduit à exprimer que la "charge" de l'unité centrale est inférieure à celle du canal de pagination (cf. Rousset de Pina [73]). Dans notre cas, elle se réduit à

$$e(\frac{M}{n}) < T \quad \text{ou} \quad \frac{M}{n} < s_T \text{ ou encore } n > n_T$$

Le taux d'activité de l'unité centrale est alors $A_n = \frac{e(\frac{M}{n})}{T}$.
Lorsqu'on ajoute un usager la variation relative de ce taux peut être définie par le rapport

$$r_n = \frac{A_{n+1}}{A_n} = \frac{e(\frac{M}{n+1})}{e(\frac{M}{n})} .$$

Ce rapport est inférieur à l'unité car e(s) est croissant. Nous définissons l'écroulement du système par un rapport nettement inférieur à 1.

Faisons l'étude de la variation de ce rapport pour les fonctions de durée de vie suivantes :

i) $e(s) = as^k$, proposée par Belady & Kuehner [9] après expérimentation pour $1.5 < k < 2.5$ et reprise par Saltzer [74] pour $k = 1$

ii) $e(s) = \frac{2b}{1+\frac{c^2}{s^2}}$, proposée par Chamberlin [23] parce que cette fonction est d'abord convexe puis concave et reste bornée,

iii) $e(s) =$ si $s \geq s_1$ alors $k'e^{ds}$ sinon 0, que nous proposons au vu des courbes* que nous assimilons à des droites sauf pour des valeurs extrêmes de s. Lorsque $s \leq s_1$ l'efficacité du système est nulle et l'étude de r_n n'a pas d'objet. En pratique s_1 est voisin de cinq pages.
Posons $n_1 = \frac{M}{s_1}$

* Figures 5.2. page 72.

On considèrera que n peut prendre des valeurs non entières pour simplifier la discussion.

En étudiant le signe de la dérivée de n pour les formes ci-dessus, on obtient le tableau de variation suivant qui ne tient pas compte de la contrainte $n \geqslant n_T$ dont nous discuterons ensuite l'influence.

e(s)	r_n	n:0 $\quad n_2 = -\frac{1}{2} + \frac{1}{2}\sqrt{1+4\frac{M^2}{C^2}} \quad n_1 \quad \infty$	comportement de r_n à l'infini
as^k	$(1-\frac{1}{n+1})^k$	0 ⟶ 1	$1 - \frac{k}{n} + 0(\frac{1}{n})$
$\frac{2b}{1+\frac{C^2}{s^2}}$	$\frac{M^2+C^2n^2}{M^2+C^2(n+1)^2}$	$\frac{M^2}{M^2+C^2}$ ⟶ $\frac{n_2}{n_2+1}$ ⟶ 1	$1 - \frac{2}{n} + 0(\frac{1}{n})$
si $s \geqslant s_1$ alors $k'e^{ds}$ sinon 0	si $s \geqslant s_1$ alors $e^{-d\frac{M}{n(n+1)}}$ sinon indefini	0 ⟶ $e^{-d\frac{s_1^2}{M+s_1}}$	indéfini

Tableau a.3.1.

Le minimum de r_n assujetti à la contrainte $n > n_T$ sera donc :

e(s)	minimum de r_n
as^k	$(\frac{n_T}{n_T+1})^k$
$\frac{2b}{1+\frac{C^2}{s^2}}$	si $n_T \leqslant n_2$ alors $\frac{n_2}{n_2+1}$ sinon $\frac{M^2+C^2n_T^2}{M^2+C^2(n_T+1)^2}$
$k'e^{ds}$	si $n_T \leqslant n_1$ alors $e^{-d\frac{M}{n_T(n_T+1)}}$ sinon indéfini

Tableau a.3.2

Exemple numérique :

$M = 10$, $C = 20 \Rightarrow n_2$ voisin de $\frac{M}{C} = 5$

$n_T = 5$

$d = 1.2$ et $s_1 = 5$ ($\Rightarrow n_1 = 20 > n_T$)

$k' = 10$

donnera :

e(s)	minimum de r_n
as^k	$(\frac{5}{6})^k$ soit entre 0.63 (k = 2.5) et 0.82 (k = 1)
$\frac{2b}{1+\frac{C^2}{s^2}}$	$\sim \frac{5}{6} = 0.82$
$k'e^{ds}$	$e^{-0.4} \sim 0.67$

Tableau a.3.3.

Lorsque l'unité centrale est saturée ($n < n_T$) on peut décrire simplement ce que vaut r_n ou B_n, taux d'activité du canal de pagination. Le tableau ci-dessous intègre ces résultats :

n	2	n_0	n_T	∞
r_n	1	1	(tableau a.3.1.)	
A_n	1	1	$\frac{e(s)}{T}$ (décroissant)	
B_n	$\frac{T}{e(s_0)}$	$\frac{T}{e(s)}$ (croissant)	1	

Tableau a.3.4.

Remarque :

L'hypothèse T constant n'est pas essentielle pour notre étude. En effet, si T est une fonction décroissante de n, on est assuré qu'il n'existe qu'une valeur de n pour laquelle $s(\frac{M}{n}) = T_n$ (rappelons qu'on considère $n \in R$).

L'étude de $r_n = \frac{e(\frac{M}{n+1})}{e(\frac{M}{n})} \times \frac{T_n}{T_{n+1}}$ peut se conduire de la même manière si la fonction T_n est connue. On remarque que r_n ainsi défini est supérieur à celui de notre étude : l'écroulement sera moins important. La forme de T_n définira comment les taux d'activités du type $\frac{T_n}{e(\frac{M}{n})}$ ou $\frac{e(\frac{M}{n})}{T_n}$ varient.

b. Etude de l'allocation d'espace maximisant l'activité de l'unité centrale

Lorsque le maximum de $F = \sum_{i=1}^{n} e_i(s_i)$ est strictement inférieur à NT_N, où $e_i(s_i)$ est le temps entre défaut de page de la tâche i qui a s_i pages en mémoire, le canal de pagination du système précédent est saturé (à l'exception près où il existerait i_0 tel que $e_{i_0}(s_{i_0}) > (N-1)T_N$; voir Rousset de Pina [73]).

Le taux d'activité de l'unité centrale est alors $\frac{\sum_{i=1}^{n} e_i(s_i)}{NT_N}$.

Le problème de maximiser cette activité est alors celui de maximiser $\sum_{i=1}^{N} e_i(s_i)$ sous la contrainte $\sum_{i=1}^{N} s_i = M$. Nous allons montrer que si toutes les tâches sont semblables $e_i(s) = e(s)$ $\forall i = 1...N$, et que e est soit concave, soit convexe, soit concave croissante puis constante, soit convexe croissante puis constante, on exprime facilement la solution.

Dans le cas de la concavité, elle se réduit à accorder aux tâches le même espace et dans le cas de la convexité à accorder le maximum (utile) à certaines d'entre elles. Ce résultat montre que si le modèle $e(s) = as^k$ était convenable, il faudrait appliquer cette dernière politique.

Nous appellerons fonction strictement convexe (respectivement strictement concave) sur [a,b] R, une fonction telle que

$a',b' \in [a,b]$ et $\lambda \in]o,1[$ on ait

$f(\lambda a'+(1-\lambda)b') \; < \text{(resp.} >) \; \lambda f(a')+(1-\lambda)f(b')$

Cette notion nous sert à séparer le cas linéaire de la convexité (respectivement de la concavité) et à assurer l'unicité des solutions du problème de maximisation.

Démontrons le lemme élémentaire suivant :

Lemme 1 :

si f est strictement $\begin{matrix}\text{convexe}\\ \text{concave}\end{matrix}$ alors $f(a)+f(b) \underset{>}{<} f(a')+f(b')$

$\forall$ a,b,a',b' tels que $a+b = a'+b'$ et $\max(a,b) > \max(a',b')$

Démonstration :

supposons $a > b$ et $a' \geqslant b'$ qui entraînent $a > a' \geqslant b' > b$

appelons Δ la relation $<$ si f est strictement convexe, $>$ si f est strictement concave. Pour les points "intermédiaires" a',b' on a :

$$f(a') \; \Delta \; \frac{a'-b}{a-b} f(a) + \frac{a-a'}{a-b} f(b)$$

$$f(b') \; \Delta \; \frac{b'-b}{a-b} f(a) + \frac{a-b'}{a-b} f(b)$$

en ajoutant membre à membre :

$$f(a') + f(b') \; \Delta \; f(a) + f(b)$$

car $2a - a' - b' = a' + b' - 2b = a - b$

Conséquence :

le maximum de $f(s) + f(m-s)$, $s \in [o,m]$ est atteint si et seulement si

$s = \frac{m}{2}$ si f est strictement concave

$s = 0$ ou m si f est strictement convexe.

La proposition suivante généralise cette conséquence.

Proposition 1

Le maximum de $F = \sum_{i=1}^{N} e(s_i)$ avec la contrainte $\sum_{i=1}^{N} s_i = M$ est atteint si et seulement si :

a. $s_i = \frac{M}{N}$ $\forall i = 1,...N$ si e est strictement concave

b. ($s_i = 0$ $\forall i \neq i_0$ et $S_{i_0} = M$) $\forall i_0 = 1,...N$ si e est strictement concave.

Démonstration (par l'absurde)

a. est équivalent à dire que tous les s_i sont égaux. Or si s_1 et s_2 étaient différents au maximum on pourrait améliorer F en remplaçant s_1 et s_2 par $\frac{s_1 + s_2}{2}$ car $2e(\frac{s_1 + s_2}{2}) > e(s_1) + e(s_2)$ d'après le lemme 1, ce qui serait absurde.

b. est équivalent à dire qu'un seul des s_i est non nul. Or si s_1 et s_2 étaient non nuls au maximum, on pourrait améliorer F en remplaçant s_1 et s_2 par 0 et $s_1 + s_2$ car $e(0) + e(s_1 + s_2) > e(s_1) + e(s_2)$ d'après le lemme 1, ce qui serait absurde.

On peut formuler une proposition semblable pour les fonctions strictement convexes (respectivement strictement concaves), croissantes jusqu'à s_0, puis constantes au-delà, que nous appelerons convexes tronquées (respectivement concaves tronquées).

Lemme 2

si e est strictement croissante jusqu'à s_0 puis constante, la solution $(s_i)_{i=1,..N}$ maximisant $F = \sum_{i=1}^{N} e(s_i)$ avec la contrainte $\sum_{i=1}^{N} s_i = M$, est telle qu'il n'existe pas deux indices i et j tels que $s_i < s_0 < s_j$.

Démonstration (par l'absurde)

S'il existe deux indices i et j tels que $s_i < s_0 < s_j$, en appelant $h = \min(s_j - s_0, s_0 - s_i) > 0$ on a $s_j - h \geqslant s_0$ et $s_i < s_i + h \leqslant s_0$. On a donc $e(s_j) = e(s_j - h) = e(s_0)$ et $e(s_i) < e(s_i + h)$ soit, en ajoutant terme à terme :

$$e(s_j) + e(s_i) < e(s_j - h) + e(s_i + h)$$

On peut donc améliorer F en remplaçant s_i et s_j par $s_i + h$ et $s_j - h$, ce qui est absurde.

On remarque que la convexité, concavité, n'interviennent pas dans ce lemme.

Conséquence

(1) les solutions vérifient donc soit $(\forall i)(s_i \leqslant s_0)$, soit $(\forall i)(s_i \geqslant s_0)$. Ces cas entraînent respectivement soit

$$\sum_{i=1}^{N} s_i = M \leqslant Ns_0 \quad \text{soit} \quad \sum_{i=1}^{N} s_i = M \geqslant Ns_0$$

(2) En excluant le cas $M = Ns_0$ pour lequel la solution unique est $(s_i = s_0)(i = 1,\ldots N)$ on obtient la réciproque :

si $M < Ns_0$ alors les solutions vérifient $s_i \leqslant s_0$ pour tout i
si $M > Ns_0$ alors les solutions vérifient $s_i \geqslant s_0$ pour tout i.

Le second cas correspond au cas trivial où tous les N-uples (s_i) satisfaisant à $s_i \geqslant s_0$ sont solutions.

Nous nous intéresserons donc au premier cas qui nécessite le lemme suivant :

Lemme 3 :

le maximum de $e(s_1) + e(s_2)$ avec $s_1 + s_2 = m < 2s_0$ est atteint si et seulement si

a. $s_1 = s_2 = \frac{m}{2}$ si e est concave tronquée

b. $s_1 = s_0$ et $s_2 = m - s_0$ (ou réciproquement) si e est convexe tronquée.

Démonstration :

d'après la conséquence 2 du lemme 2 nous avons forcément $s_1 \leqslant s_0$ et $s_2 \leqslant s_0$. Le lemme 1 nous donne immédiatement les résultats a. et b., dans lequel la contrainte $s_1 \leqslant s_0$ et $s_2 \leqslant s_0$ limite l'écart que peuvent prendre les variables.

De là nous pouvons déduire la proposition suivante :

Proposition 2 : si $M < Ns_0$

La solution $(s_i)_{i=1,..N}$ maximisant $F = \sum_{i=1}^{N} e(s_i)$ avec la contrainte $\sum_{i=1}^{N} s_i = M$ est :

a. $\forall i = 1,...N \quad s_i = \frac{M}{N}$ si e est concave tronquée

b. en posant n_0 = partie entière de $(\frac{M}{s_0})$

$(s_i = s_0 (\forall i = 1,...n_0),\ s_{n_0+1} = M - n_0 s_0, s_i = 0 (\forall i > n_0+1))$

et toutes les permutations, si e est convexe tronquée.

Démonstration :

comme $M < Ns_0$, on doit avoir $s_i \leqslant s_0$

a. si, au maximum s_1 et s_2 étaient différents en les remplaçant par $\frac{s_1 + s_2}{2}$ on pourrait d'après le lemme 3 améliorer F ce qui serait absurde.

b. si, au maximum, s_1 et s_2 étaient à la fois différents de 0 et de s_0, donc $s_1 + s_2 < 2s_0$, on pourrait d'après le lemme 3 améliorer F en les remplaçant par s_0 et $s_1 + s_2 - s_0$, ce qui serait absurde.