UNIVERSITÉ DE
SHERBROOKE

Département d'informatique
Faculté des sciences

IFT 630 - Processus concurrents et parallélisme

Chapitre 10

Performance

Gabriel Girard[1]

Sherbrooke

18 octobre 2022

[1] Gabriel.Girard@usherooke.ca

Table des matières

Chapitre 1

Performance

L'objectif primordial de tout système informatique est l'accomplissement des spécifications pour lesquelles il a été conçu. Dans bien des cas, le second aspect d'intérêt majeur privilégié est une performance adéquate à un coût raisonnable. [1]

Un exemple de la vie courante sert à illustrer ces différents besoins. Si vous désirez acheter un véhicule, vous le choisissez d'abord pour qu'il soit fonctionnel selon les buts visés. Ainsi, si l'utilisation principale du véhicule est :

- la promenade $\longrightarrow$ voiture berline ;
- transport d'une équipe de baseball $\longrightarrow$ autobus ;
- transport de bois $\longrightarrow$ camion ;
- transport de marchandise $\longrightarrow$ camion ;
- etc.

Fréquemment, la deuxième particularité recherchée par le consommateur sera un élément de performance (consommation, accélération, puissance, vitesse maximale). [2]

Comme pour les véhicules, le choix d'un logiciel (ou système informatique) dépend de plusieurs facteurs, l'un des plus importants étant celui de la performance. Le poids de chacun de ces facteurs varie selon les personnes et le type d'application.

Le système d'exploitation étant le gestionnaire des ressources (le moteur du véhicule), il est nécessaire de déterminer, pour ses concepteurs, l'efficacité de la gestion d'un système particulier. De même, tout logiciel que l'on installe affecte la performance de notre environnement dans sa globalité. Il existe donc différentes techniques pour mesurer l'efficacité avec laquelle un système ou un logiciel effectue le traitement demandé.

De plus, sur une installation particulière, de grandes améliorations sont généralement possibles au niveau de l'utilisation des ressources (mise au point sur une voiture). Pour cela, une surveillance et une évaluation continues s'avèrent judicieuses. Malheureusement, la plupart des installations en font peu de cas et, quand elles s'en préoccupent, cela génère une masse de données que peu savent interpréter. Il est donc capital d'exercer une surveillance autant que d'être apte à en tirer les résultats afin de procéder à «la mise au point» du système.

L'évaluation de performance exige donc de vastes connaissances qui vont de la conception et de

1. Selon les buts visés du système, le second aspect souhaité peut être la fiabilité, la sécurité ou tout autre aspect.
2. Cela peut aussi être l'apparence, la résistance, la sécurité...

l'implantation d'instruments de mesure à l'analyse mathématique de modèles de file d'attente en passant par une réelle expertise des systèmes à évaluer.

Mais il y a aussi un principe auquel on ne peut échapper lorsque l'on analyse un environnement logiciel. Il s'agit du principe d'Heisenberg qui s'énonce comme suit :

> *On ne peut pas mesurer un système sans l'affecter d'une façon quelconque.*

Évidemment, certains outils de mesure ont moins d'impact que d'autres. Donc, il s'agit aussi d'être conscient de leurs effets sur l'environnement évalué.

Dans ce chapitre, nous allons ainsi étudier les concepts de performance afin de :

1. reconnaître les bienfaits de l'évaluation de performance ;
2. connaître les éléments de base qui vont de pair avec une telle évaluation.

1.1 Tendances

Les progrès technologiques affectent énormément les tendances pour évaluer la performance. Ainsi, il y a quelques années (30 ans et +), l'accent était mis sur l'ordinateur. On le voulait performant sans toutefois s'entendre sur les paramètres à utiliser pour conclure. Évidemment, on considère normalement la vitesse des différentes unités. Cependant, pour bien estimer la «capacité» (ou puissance) d'un ordinateur, on doit également tenir compte de l'organisation de la machine, de la dimension des mots, de la dimension du «bus», du nombre d'adresses par instruction, etc.

Un peu plus tard, on a mesuré la performance en considérant la productivité humaine. Cela était probablement dû à l'augmentation du coût de la main d'œuvre et du temps requis pour le développement d'un logiciel (importance en augmentation).

Avec l'apparition des micro-processeurs (années 80), on s'est attardé à la performance des unités d'entrées/sorties, des canaux et autres périphériques, alors qu'avec celle des ordinateurs personnels, où l'importance de la puissance de l'UCT et des unités d'entrées/sorties comptent moins, on a mis l'accent sur la disponibilité. En même temps que les ordinateurs personnels, les réseaux d'ordinateurs sont apparus. Pour ces systèmes, les mesures de performance sont nombreuses : latence des communications, vitesse de transmission, répartition des ressources, efficacité des serveurs, performance de la mémoire tampon (cache), ...

Finalement, avec la baisse du coût d'achat des ordinateurs, le coût et la performance des logiciels sont devenus très importants. En effet, le logiciel, qui crée une machine virtuelle, peut causer des ralentissements significatifs même sur les ordinateurs les plus puissants. Il est donc primordial, sur les systèmes modernes, de contrôler la performance du logiciel aussi bien que celle du matériel. Dans ce dernier cas, les capacités du système d'exploitation sont cruciales à la performance du système, tout particulièrement les caractéristiques de multiprogrammation et de multitraitement. De plus, certains logiciels (applications, communications, web, bibliothèques) ont tendance à consommer à l'excès les ressources du système. Pensons à Teams qui, sur Linux, consomme plus de 1Gig de mémoire en plus de créer une armée de processus. Firefox et Thunderbird sont aussi des logiciels qui consomment énormément de ressources. Des outils, tels les compilateurs, génèrent parfois du code qui affecte aussi la performance du système (des optimisations à ce niveau sont possibles). Ce sont tous des éléments majeurs dont la performance affecte grandement celle de l'environnent dans sa globalité.

1.2 Besoins pour l'évaluation de performance

En général, quatre raisons justifient une évaluation de la performance :

1. **La sélection** ;

   L'évaluation de performance d'un ordinateur et de ces logiciels en évidemment requise en prévision d'un achat.

2. **La projection** en vue de la conception d'un nouveau système ;

   Précédant la phase même du développement d'un nouvel ordinateur, d'un nouveau logiciel ou d'un nouveau composant matériel (disque, réseau, ...), on doit réfléchir aux performances attendues par celui-ci (ou ceux-ci).

3. **Le contrôle** ;

   Le contrôle de performance est la collection et l'analyse d'information sur la performance d'un système existant. On accumule des données sur la performance du système actuel ou d'un de ses composants afin :

   (a) de s'assurer que le système rencontre ses objectifs de performance. Le contrôle détermine la façon dont un système opère en terme de rendement, temps réponse, prévisibilité, etc.

   (b) d'aider à estimer l'impact de certains changements planifiés (reconfiguration, ajout de composants ou de logiciels),

   (c) de fournir à l'administration les données dont il a besoin pour prendre des décisions stratégiques. Ainsi, un administrateur informé du résultat de certaines mesures peut décider de modifier les priorités existantes concernant les travaux du le système. Le contrôle de performance renseigne aussi les personnes en charge sur la distribution des divers types de travaux. S'il est déterminé que la plupart des travaux qui s'exécutent sont des programmes de production, alors l'usage de compilateurs optimisant est recommandé pour générer du code plus efficace pour ces programmes. Si, au contraire, la plupart des travaux concerne le développement et la mise au point de logiciels, alors le recours à des options de compilations rapides est plutôt à privilégier (*quick and dirty*).

   Le contrôle de performance fournit à la personne en charge de l'environnement des indications précises sur le fonctionnement du système et cette information est particulièrement importante dans un environnement pour lequel des décisions clés de conception et de configuration sont prises ou modifiées sur la base des opérations observées du système.

   (d) de détecter, de localiser et d'éliminer la ou les sources de certains problèmes de performance tels les goulots d'étranglement, l'exemple typique. Les goulots d'étranglement se détectent par la surveillance des files de requêtes pour chaque ressource. Lorsque la taille de la file croit rapidement, c'est que le taux d'arrivée est supérieur au taux de service. Les traces des instructions ou des modules exécutés aident également à détecter des goulots d'étranglement. En particulier, celles des modules peuvent révéler qu'un petit sous-ensemble d'entre eux monopolise la majorité du temps de traitement (en %). Les concepteurs sot alors en mesure de concentrer leurs efforts d'optimisation sur ces modules et ainsi améliorer considérablement la performance du système et ce sans dépenser efforts et ressources sur des parties peu ne nécessitant aucune amélioration.

   Le retrait des goulots d'étranglement est une partie importante de la «mise au point» d'un logiciel. Ils doivent être éliminés en augmentant la capacité des ressources ou en ajoutant d'autres ressources du même type.

Exemple d'options pour l'optimisation

Les compilateurs pour le langage C++ (g++ et Clang) offrent plusieurs niveaux d'optimisation [3, 2] :

- (-O0) : aucune optimisation (le niveau le plus bas et le défaut) ;
  Ce niveau représente la meilleure option pour le développement de logiciel car la compilation est rapide.
- (-O1) : optimisation minnimale ;
- (-O2) : optimisation moyenne ;
  Par exemple, g++ intègre les petites fonctions directement dans le code (inline).
- (-O3) : optimisation optimale ;
  Ce niveau représente la meilleure option pour les logiciels de production car on génère le code le plus efficace possible. Elle intègre le lus de fonctions possibles directement dans le code (inline) et utilise des instructions de traitement parallèle (*loop vectorization* et instructions de type SIMD). La taille du code généré est toutefois plus significative et le temps de compilation est très long.
- (-OS) : optimisation en vue de réduire la taille du code généré ;
  Similaire au niveau 2 avec des optimisations pour réduire la taille du code.
- (-Oz) : optimisation encore plus poussée sur la taille du code (Clang seulement) ;
- (-Ofast) : optimisation sans regard à la conformité au standard ;
- (-Og) : optimisation pour la mise au point.
  Ce niveau est similaire à O1 mais certaines optimisation seront éliminées pour améliorer la facilité de mise au point.

Les goulots d'étranglement

Un système d'exploitation ou tout logiciel qu'il héberge gèrent une collection de ressources (matérielles ou logicielles). Celles-ci interagissent de façon complexe et ainsi affectent les opérations de tout l'environnement. Occasionnellement, une ou plusieurs ressources deviennent des goulots d'étranglement, ce qui limite la performance globale du système car celles-ci ne parviennent pas à faire leur part de travail. Une ressource constitue un goulot d'étranglement si elle ne peut faire les traitements demandés par d'autres ressources assez rapidement pour les «tenir occupées» et cela même si ces dernières sont sous-utilisées.
Un goulot d'étranglement se développe à une ressource lorsque le traffic des requêtes, des tâches ou des processus à cette ressource approche de sa capacité maximale de traitement. À ce point, on dit qu'une ressource devient saturée, ce qui signifie que les processus en compétition pour cette ressource commence à interférer les unes avec les autres.
Un exemple classique de saturation dans les systèmes d'exploitation est celui de l'écroulement. Ce phénomène se produit dans un système paginé lorsque la mémoire principale est trop sollicitée et que les ensembles de travail des processus actifs ne peuvent être maintenus simultanément en mémoire.

4. **La planification de capacité**.
   Une raison majeure pour justifier une évaluation de performance est certainement la planification de la capacité du système dont le principal but est de déterminer la fraction de la capacité totale du système occupée par la charge actuelle. Cette information permet de prédire la capacité nécessaire pour traiter dans le futur la charge espérée tout en conservant le niveau de service requis [3].
   Certains outils sont conçus pour se charger de la planification de la capacité. Par exemple, l'outil Best/1 de BGS Systems Inc. [1, 4] prédit le comportement d'un système sous différentes conditions telle que l'augmentation du nombre d'usagers. Appliqué de cette façon, il permet de faire de la planification de capacité.
   Il ne faut pas confondre la planification de capacité avec l'estimation de l'impact de changements planifiés (point 3b du contrôle). En fait, l'un constitue l'inverse de l'autre. Soit une configuration C et une charge L. La performance P du système est définie par

$$P = C(L).$$

   Dans le cas du contrôle, on tente d'estimer l'impact d'une modification sur la configuration matérielle ($C \Rightarrow C'$). Généralement, on espère obtenir une performance

$$P' = C'(L) \text{ tel que } P' \geq P.$$

   Dans le cas de la planification, on cherche à savoir si la configuration actuelle (C) permet de supporter une nouvelle charge (L'). Généralement, on obtient une performance

$$P' = C(L')$$

   qui contribue à établir si la configuration actuelle supportera cette nouvelle charge.

Besoins en évaluation pendant toutes le phase de développement

L'évaluation de performance est nécessaire dès le début de la conception d'un nouveau logiciel, dans les opérations journalières du celui-ci et dans toutes considérations de modifications ou de remplacements.

La **prédiction** est à la base de toute évaluation de performance. Dans les premiers temps, la personne en charge du développement tente de prédire :

- la nature des applications qui s'exécuteront sur le système ;
- la charge de travail anticipée pour ces applications.

Lorsque qu'une personne initie le développement, l'implantation ou la configuration d'un nouveau logiciel (ou système), elle met à profit l'évaluation de performance et la prédiction pour déterminer :

- la meilleure organisation matérielle ;
- la stratégie de gestion des ressources qui devrait être implantée dans le nouvel environnement (système d'exploitation, applications, ...) ;
- si le logiciel en développement rencontre ses objectifs de performance.

3. Cela requiert aussi une autre démarche soit celle qui consiste à prédire la charge future.

Une fois le logiciel mis en production ou commercialisé, la personne en charge de sa mise en production doit répondre aux diverses questions des usagers sur les performances du système face à certains environnements. Des évaluations devront donc avoir été réalisées afin de fournir ces résultats.

Chaque logiciel vendu ou installé doit être configuré pour satisfaire chacune des clientèles. Lors de cette étape, là encore, des évaluations de performance sont requises pour procéder à des ajustements (ou optimisations) si les résultats en démontrent la nécessité.

Une fois le logiciel en place chez la personne utilisatrice, cette dernière et le fournisseur cherchent à obtenir la performance optimale. Ainsi, d'autres ajustements sont à prévoir afin d'obtenir la meilleure performance possible dans cet environnement particulier. On appelle cette étape la **mise au point du système** (*fine tuning*). Des augmentations significatives de la performance d'un logiciel étant possibles lorsque celui-ci est ajusté aux besoins spécifiques de l'environnement, notre étude portera principalement sur ce dernier type d'évaluation de performance.

Le gain de performance obtenu par une mise au point fine du logiciel s'avère parfois difficile à quantifier. Par exemple, il est complexe de mesurer :

- un degré accru de satisfaction de la personne qui utilise ce système (suite à un meilleur temps réponse) qui entraîne possiblement une augmentation de sa productivité (accélérant ainsi le développement de nouvelles applications) ;
- une augmentation de clientèle suite à un meilleur traitement des requêtes.

Ceci dit, l'impact de certains actions se quantifie plus aisément, comme la réduction du temps d'opération du système, le décroissement du coût de la configuration (moins de canaux ou de disques), l'augmentation de la durée de vie de l'environnement actuel (avant son remplacement dû à une charge trop lourde ou un temps réponse trop lent).

Avant d'entreprendre une étude de performance, on se doit de considérer les coûts associés à cette évaluation versus les bénéfices espérées. Ces estimations dépendent de la condition de tout l'environnement par rapport à sa condition optimale, qui elle n'est pas connue. Plus on est près de cette condition optimale moins le gain sera grand.

La procédure typique pour optimiser le ratio coût/performance est illustrée au diagramme de la figure 1.1. Cette procédure est itérative. Elle contient un cycle comprenant des phases de diagnostics et de thérapies. Le but de la phase diagnostique est de déterminer la cause d'une mauvaise performance (un goulot d'étranglement ?). Selon la ou les conclusions, différentes thérapies sont envisageables mais évidemment on choisit généralement celle qui augmente le plus la performance et, ce, au moindre coût.

1.3 Mesures de performance

Pour discuter de performance, définissons clairement ce terme. Par performance, nous entendons la manière dont et comment, en terme d'efficacité, un système informatique rencontre ses objectifs.

Définition : Performance

La performance est définie comme la manière dont et comment, en terme d'efficacité, un système informatique rencontre ses objectifs.

Cette identification n'est qu'un pas initial vers une définition significative. En effet, l'évaluation

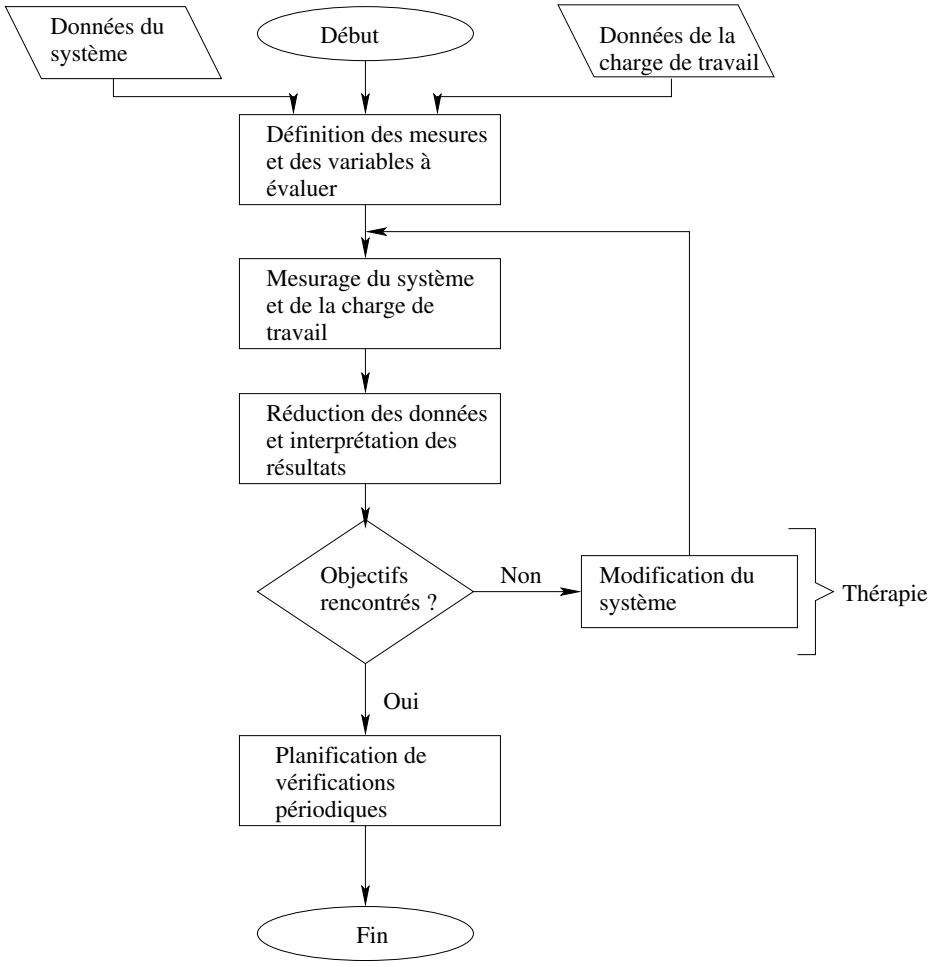

Figure 1.1 – Procédure pour le contrôle de performance

de performance d'un système informatique intéresse des catégories fort variées d'individus, chacun d'eux donnant une signification particulière au mot performance. Nous décrivons donc ces catégories et leurs points de vue avec comme objectif de raffiner notre définition de performance.

Ici, on regroupe les personnes confrontées à des problèmes d'évaluation selon trois classes :

1. les personnes en charge de la conception du système informatique ;
2. les gestionnaires d'installation informatique ;
3. les personnes qui utilisent le logiciel.

Même s'ils ont le même but (i.e. rendre les opérations du système efficace), les trois classes considèrent le problème sous des angles différents. Alors que les personnes en charge de la conception du logiciel ou du système (matériel et logiciel) doivent penser à toutes les utilisations possibles du système, les gestionnaires, eux, doivent appréhender les besoins d'un environnement spécifique. Les

personnes utilisatrices, elles, ne sont généralement intéressées qu'à l'exécution de leurs programmes.

Partant du concepteur vers les usagers, l'étendue des exigences à rencontrer se restreint et les possibilités d'influencer la performance deviennent de plus en plus limitées.

Les trois classes emploient des variables spécifiques pour exprimer leur vision de la performance et, dans certains cas, assignent des poids différents aux mêmes variables :

- Le concepteur d'un système est généralement préoccupé par l'utilisation des composants du système. Il peut influencer cette mesure en jouant avec certaines variables tel le temps d'accès à la mémoire, la vitesse du matériel, l'organisation des bibliothèques de programmes et des bases de données, les algorithmes d'allocation de la mémoire, etc. Les autres variables, tel le temps d'exécution d'un programme, ne sont que de peu d'intérêt pour le concepteur.
- Le gestionnaire de l'installation sera plus intéressé par un usage bien balancée et rentable des composants du système, par un service satisfaisant à la plupart de sa clientèle (ou aux plus importantes) et aussi par un prix équitable pour l'utilisation des facilités. Par exemple, pour un gestionnaire, plus il y a de personnes qui utilisent sont environnement, plus c'est payant. Il sera donc heureux du grand nombre d'usagers supporté.
- Un usager influence la performance en changeant la charge sur les ressources (UCT, périphériques, mémoire et canaux) et évalue l'efficacité du système en terme de temps d'exécution et de coûts. Les variables des autres classes ne le concernent guère.

Les points de vue subjectifs (personnels et partiels) sur la performance du système qui caractérise les membres de chaque classe sont rendus plus objectifs en faisant appel des variables précises qui décrivent certains aspects spécifiques de la performance globale du système. Plusieurs variables sont difficilement mesurables ou même ne sont pas quantifiables du tout. Par exemple, la facilité d'utilisation, la puissance d'un ensemble d'instructions sont des variables difficilement mesurables. D'autres variables, tel le nombre d'accès au disque par minute, sont beaucoup plus faciles à quantifier. En général, un évaluateur doit tenir compte de ces deux types de variables. Toutefois nous n'abordons ici que les variables qui peuvent être mesurées.

Il est important de toujours se souvenir qu'une évaluation de performance n'est jamais absolue. En effet, celle-ci est toujours prises pour servir de base de comparaison. Pour bien effectuer cette comparaison, toute évaluation de performance requiert toujours les informations suivantes :

1. les caractéristiques du système évalué ;

   Dans le cas d'un système informatique, ces variables incluent les données sur la configuration du matériel et du logiciel du système (ex. : taille de la mémoire, nombre de canaux et de disques, localisation des systèmes de fichiers) et sur les caractéristiques des différents composants (UCT, type et vitesse des canaux, temps d'accès au disque, etc).

   Ces données sont vitales lorsque l'on veut comparer la performance de différents systèmes. Les données sur la configuration prennent beaucoup plus d'importance lorsque l'optimisation d'un seul système est en cause. [4]

   Lorsque l'évaluation vise un sous-système, les mêmes variables peuvent servir à caractériser le sous-système visé plutôt que le système complet. Cependant, cela est possible seulement si le sous-système ne subit aucune influence externe. Par exemple, lorsque l'on détermine le placement des fichiers sur disques pour minimiser le mouvement de la tête de lecture, le sous-système comprend le canal, le contrôleur et le disque.

4. On doit, en effet, comparer la performance des différentes configurations entre elles.

2. les conditions d'opération du système au moment de l'évaluation ;

   Les conditions d'opérations du système consiste en la description de la charge de travail présente lorsque l'évaluation a été faite. La charge de travail d'un système est composée des programmes, des données et des commandes que l'usager soumet au système.

   On ne peut évaluer un système sans étudier les caractéristiques de sa charge de travail. En effet, la performance globale du système peut être modifiée de façon significative par des charges différentes sur les composants du système.

   La description de la charge de travail et sa caractérisation sont des problèmes difficiles et délicats. Une des approches les plus populaires est l'approche probabiliste. On représente la charge, dans ce cas, par des variables aléatoires dont les distributions sont indépendantes. Par exemple, un modèle simple de charge de travail pourrait être représenté par les distributions du temps entre chaque arrivée et du temps d'exécution. Chaque programme est donc évalué comme une paire de valeurs.

3. les mesures de performance du système.

   Cette classe de variables se divise en deux catégories distinctes : les mesures dites internes, qui évaluent l'efficacité de l'utilisation de chacun des composants du système, et les mesures dites externes, qui évaluent l'efficacité du traitement externe au système. Le taux d'utilisation de l'UCT et le taux de pagination sont des exemples de mesures internes tandis que le temps réponse et le temps de virement sont des exemples de mesures externes.

   Les mesures internes sont orientées vers le système et les mesures externes le sont vers les usagers. Les mesures internes sont donc celles qu'adoptent les concepteurs et les administrateurs pour évaluer un système tandis que les mesures externes sont généralement choisies par les usagers.

Notez que les variables de la classe 1) s'associent aux intrants «*Données du système*» de la figure 1.1, alor que le variables de la classe 2) s'associent aux intrants «*Données de la charge de travail*» de la figure 1.1. Les variables du points 3) correspondent à la première boîte de traitement de cette même figure.

1.3.1 Mesures communes de performance d'un système

Voici quelques mesures de performances ainsi que certaines mesures dérivées.

Le temps de virement

Le temps de virement est une mesure utilisée pour évaluer l'efficacité avec laquelle des travaux non interactifs sont traités. Elle est utilisée particulièrement sur les systèmes de traitement par lots.

On distinque deux types de temps de virement : le temps de virement externe et le temps de virement interne.

Le temps de virement externe d'un programme est l'intervalle de temps entre l'instant à partir duquel l'usager soumet le programme et l'instant où il reçoit les résultats (+ la manipulation manuelle s'il y a lieu). On l'appelle externe car cela inclut aussi le temps pour les opérations manuelles d'entrées/sorties en plus de l'exécution. Le temps requis au système pour exécuter le programme incluant la lecture et l'impression est appelé le temps de virement interne ou simplement

temps de virement. Cette mesure est donnée par $T = P - R$ où R est le moment du début de la lecture et P le moment de la fin de l'impression.

Le temps de virement externe (lorsque cela existe encore) est souvent beaucoup plus long que le temps de virement. Il est cependant important de noter que le temps de virement externe est une mesure quelque peu désuète. En effet, il est assez rare que des operations manuelles de la part d'une tierce personne soit nécessaire pour soumettre un travail ou en obtenir les résultats. Toutefois, la manipulation de rubans magnétiques par des robots peut être considérée comme une intervention manuelle.

Il existe plusieurs mesures dérivées du temps de virement : le temps de traitement, le temps de virement moyen, etc. Ces mesures se justifient par le fait que le temps de virement T ne fournit pas une information vraiment utile pour évaluer l'efficacité avec laquelle le système a traité le programme.

Exemple : Soit deux programmes A et B qui requièrent respectivement une minute et trente minutes d'exécution. Même si le temps de virement sont deux minutes et 33 minutes, on ne peut pas conclure que A a été mieux traité que B si on n'a pas d'autres informations tel le temps de traitement.

Voici donc quelques mesures dérivées qui nous permettront de mieux évaluer l'efficacité du système :

- le temps de virement réel ou temps de traitement (T_p) ;

  Cette mesure correspond au temps de virement lorsque le programme est seul dans le système. Normalement le temps de virement est différent lorsqu'il y a plusieurs usagers.

- le temps de virement moyen ;

  Cette mesure est définie comme
  $$T_m = \frac{\sum_{i=1}^{n} T_i}{n}$$
  où T_i représente le temps de virement de tous les travaux et non pas i fois le temps de virement pour le même travail.

  Cette mesure est supérieure au temps de virement mais elle peut mener à des mauvaises conclusions sur l'efficacité de traitement particulièrement si n est petit.

- le temps de virement pondéré.

  Cette mesure, notée T_w, est définie comme le ratio entre le temps de virement et le temps de traitement du programme (T_p).
  $$T_w = \frac{T}{T_p}$$
  L'utilisation de cette mesure avec un nombre *pure* rend l'efficacité du traitement de différents programmes comparables.

Le temps de virement pondéré est affecté par les politiques de gestion des ressources implantées par le système et par les caractéristiques de la charge de travail.

- le temps de virement pondéré moyen ;

  Cette mesure, notée T_{wm} est définie comme :

$$T_{wm} = \frac{\sum_{i=1}^{n} T_{w_i}}{n}$$

La figure 1.2 illustre les différents temps de virement de base.

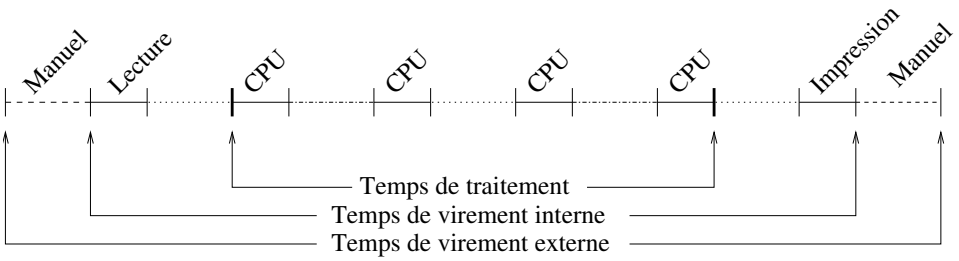

Figure 1.2 – Temps de virement

Le temps de réponse

Le temps réponse est utilisé sur les systèmes interactifs (c'est donc la contrepartie du temps de virement sur les systèmes à traitement par lots). Il est défini comme l'intervalle de temps entre l'envoi d'une commande (return) au terminal et l'instant ou le terminal commence à afficher la réponse du système. Ce temps peut être mesuré par différents outils matériels ou logiciels.

On appelle le temps de réponse **dur** la somme de toutes les tranches de temps utilisées par un processus.

Le temps réponse dépend de la commande. Il y a donc une distinction entre les commandes lourdes et légères. Les commandes légères utilisent normalement moins d'une tranche de temps pour s'exécuter. Par exemple, les commandes de l'éditeur du type insertion, destruction ou modification d'un texte, et les commandes login, time et date sont des commandes légères.

Les commandes lourdes requièrent plus d'une tranche de temps. Par exemple, la compilation, l'exécution, l'assemblage et le tri sont souvent des commandes lourdes.

Le temps réponse, tout comme le temps de virement, n'est pas une mesure très significative. On utilise donc fréquemment des mesures dérivées. Voici donc quelques mesures dérivées :

- le temps de réponse moyen ;

  Le temps réponse moyen, au même titre que le temps de virement moyen, n'est pas une mesure qui fournit une information complète sur la performance d'un système interactif. Par exemple, il ne fournit aucune information sur la stabilité de la performance, i.e. que les valeurs prises pour la moyenne peuvent être identiques ou très variables.

  On doit donc trouver une autre mesure pour vérifier la distribution des temps réponses. Cette mesure donnera plus d'information sur la stabilité du système. Il peut, en effet, y avoir un

grand écart entre les temps réponses des usagers sur un système avec une charge légère et avec une lourde charge.

La moyenne étant trop dépendante des valeurs choisies, on utilise, par exemple, l'écart-type, la variance, la médiane, qui sont des bons descripteurs de la distribution pour caractériser la performance d'un système interactif.

- l'écart-type ;

  L'écart-type est :

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(R_i - R_m)^2}$$

  où R_m est le temps réponse moyen et chaque R_i un temps réponse de la $i^{ième}$ commande.

  L'écart-type σ donne quelques indications de la viabilité du temps réponse ou du degré de dispersion autour de la valeur moyenne. Ainsi, si σ est petit alors le système est stable. Au contraire, s'il est grand, on devrait réévaluer le système qui est soit surchargé soit insuffisant.

- l'étendue ;

  Cette mesure est la différence entre les deux temps réponses extrêmes. Elle aussi donne une idée de la stabilité du système.

- la variance ;

  La variance est σ^2 et est équivalente a σ car elle fournit la même information. Une petite variance indique que les temps réponses demeurent toujours près de la moyenne. Lorsque la variance est très grande, une analyse du système devrait être faite car le système semble chargé de façon anormale ou bien pas assez puissant.

  La variance, comme l'écart-type, sont des mesures importantes pour l'usager qui aime normalement un système stable.

- la valeur maximale ;

  Les usagers évaluent souvent le temps réponse d'un système de façon subjective selon la valeur maximale car ils y sont plus sensibles. Celle-ci se produit peu souvent contrairement au minimum qu'il ne considère généralement pas.

- les centiles ;

  On défini les centiles de la façon suivante :

  Soit un ensemble de valeurs en ordre croissant ou décroissant, la valeur qui divise l'ensemble également en moitié est appelée la médiane. Les valeurs qui divisent cet ensemble en quatre parties égales sont les quartiles. Dans le premier quartile on retrouve le quart des valeurs, etc. Les centiles divisent l'ensemble en cent parties égales. La médiane est le rang centile 50. Les quartiles sont les rangs centiles 0, 25, 50 et 75.

La valeur du temps réponse et de toutes les mesures dérivées dépend de plusieurs éléments : l'efficacité du programme qui gère le terminal, les caractéristiques du terminal, les programmes d'applications, le nombre d'usagers interactifs et la charge non interactive.

Lorsque l'on parle de communication et de certains sous-systèmes, on ne parle parfois plus de temps réponse mais de latence.

Le rendement ou débit

Le principal objectif de la plupart des évaluations est l'amélioration du rendement. Le rendement peut être informellement défini comme le montant de travail fait par un système par unité de temps. Sa valeur peut être exprimée de plusieurs façons :

- nombre de programmes par unité de temps ;
- quantité de données traitées par unité de temps ;
- nombre de transactions traitées par unité de temps ;
- nombre d'entrées/sorties par unité de temps ;
- ...

Le rendement peut aussi bien s'appliquer à un système complet qu'à un composant du système. Par exemple, on peut évaluer le rendement d'un canal d'entrées/sorties ou de l'UCT.

Le rendement d'un système est généralement inférieur à la valeur théorique sur la capacité du système. Une définition du rendement R est :

$$R = \frac{N_p}{T_{tot}}$$

où N_p est le nombre de programmes (ou transactions) exécutés pendant la période de temps T_{tot}. Lorsque l'on passe un banc d'essais, N_p est le nombre de programmes dans le charge et T_{tot} est le temps d'exécution du banc d'essais. **La valeur du rendement doit toujours être accompagnée par une description de la charge qui a été évaluée**.

La valeur R donne certaines indications sur la vitesse d'exécution des programmes mais n'apporte aucune indication sur l'efficacité du traitement de chaque programme.

Le rendement étant une mesure externe il fournit une indication globale de la puissance du système mais aucune information pour l'évaluation de performance d'un seul programme.

Le rendement est influencé par les facteurs suivants :

- les caractéristiques de la charge avec laquelle l'évaluation a été faite ;
- la configuration du système ;
- le degré de multiprogrammation autorisé par le matériel ;
- les algorithmes utilisés pour l'allocation des ressources ;
- la vitesse des composants matériels et logiciels ;
- la possibilité de chevauchements des composants ;

Ces facteurs influencent plus ou moins fortement le rendement. L'importance de cette influence est mesurée par la sensibilité du rendement à la variation du facteur. La sensibilité est définie comme le ratio entre la variation de la mesure et celle du facteur. Cependant lorsque le facteur n'est pas quantitatif la variation du rendement seulement déterminera sa sensibilité.

L'importance du rendement comme mesure est due au fait que sa valeur est fortement reliée au coût d'utilisation du système.

Des exemples de mesures de rendement sont le nombre de transactions par seconde (TPS) et les SPEC (SPECint, SPECfp, SPECrate).

La capacité

La capacité est définie comme la valeur théorique maximale du rendement d'un système. En pratique, selon une certaine charge, sa valeur correspond à la quantité maximale de travail que le système est capable de faire par unité de temps (la capacité théorique).

Normalement, lorsque la charge augmente, un phénomène de saturation se produit qui empêche le système d'atteindre sa capacité théorique et qui éventuellement cause une réduction du rendement. Le point de saturation du système est la charge maximale (capacité pratique) que le système peut traiter avant que le rendement ne commence à diminuer. La capacité pratique est normalement inférieure à la capacité théorique. La figure 1.3 illustre ce phénomène.

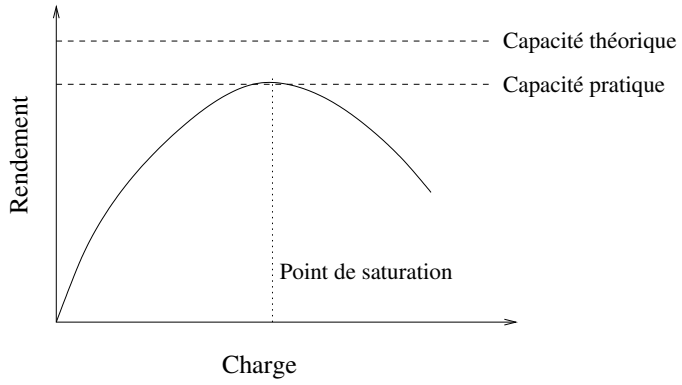

Figure 1.3 – Capacité et saturation

Dans le cas des médiums de communication, le rendement théorique maximal est appelé largeur de bande (bandwidth). Ce rendement ne tient pas compte de l'overhead causé par le protocole. Le débit maximal (ou rendement maximal) est le terme utilisé lorsque l'on considère l'overhead.

Exemple :

- FAST SCSI

  Capacité (sans overhead) = 10 Mbyte/sec.

  Débit maximal (en considérant l'overhead) = 7.5 Mbyte/sec.

- Wide SCSI

  Capacité (sans overhead) = 20 Mbyte/sec.

Débit maximal (en considérant l'overhead) = 16 Mbyte/sec.

- MBus et XBus

  Capacité (sans overhead) = 320 Mbyte/sec.

  Débit maximal (en considérant l'overhead) = 100 Mbyte/sec (MBus) et 250 Mbytes/sec (XBus).

La disponibilité

La disponibilité est définie comme le pourcentage de temps total pendant lequel le système est disponible pour les usagers.

Le temps moyen entre les pannes (MTBF)

Cette mesure est définie comme l'intervalle de temps entre deux bris du système.

L'utilisation de l'UCT

C'est le pourcentage de temps d'opération pendant lequel l'UCT est active. L'utilisation est normalement subdivisée selon la contribution de tous les types d'activités sur l'UCT.

Par exemple, L'UCT peut avoir trois états que l'on peut représenter dans son utilisation : libre, usager et superviseur. À l'état usager, l'UCT exécute un travail pour un usager qui est «chargé» à cet usager. À l'état système, l'UCT est dans le système d'exploitation. Une partie de ce temps peut être chargé à un usager et le reste est de l'«overhead», i.e. du temps pris par le système d'exploitation pour sa gestion interne. Dans certain système, cet overhead peut être considérable. Lorsqu'on mesure l'utilisation de l'UCT, on doit toujours distinguer le travail productif et chargeable à un usager de l'overhead.

L'utilisation de l'UCT est souvent utilisée comme mesure de rendement car le rendement lui est souvent proportionnel.

La mesure d'utilisation est cependant une mesure décevante. En effet, même si on désire un haut degré de multiprogrammation, cela peut résulter en une utilisation inutile. Ainsi, le meilleur moyen d'obtenir une bonne utilisation de l'UCT consiste à exćuter une série de processus qui font une boucle infinie.

Utilisation

Cette mesure est définie comme précédemment sauf qu'elle est applicable à l'ensemble de toutes les ressources.

Le chevauchement

Le chevauchement est le pourcentage de temps d'opération du système pendant lequel deux ou plusieurs ressources sont occupées simultanément. Cette mesure est significative seulement pour les systèmes multiprogrammés. Elle est particulièrement importante car elle fournit de l'information à savoir si le chevauchement des canaux d'entrées/sorties entre eux ou avec l'UCT est le maximum espéré sous la charge évaluée.

Normalement, cette mesure est divisée pour donner toutes les combinaisons possibles de l'activité des composants. Par exemple, les canaux 1, 2 et l'UCT sont occupés ou libres.

Ces mesures sont souvent représentées par des graphes de Gantt ou de Kiviat. Le graphe de Gantt représente le chevauchement en utilisant une série de lignes pour représenter (dans le temps) le temps d'utilisation de chaque composant. On indique à droite, en pourcentage, le total d'utilisation. La figure 1.4 montre un graphe de Gantt.

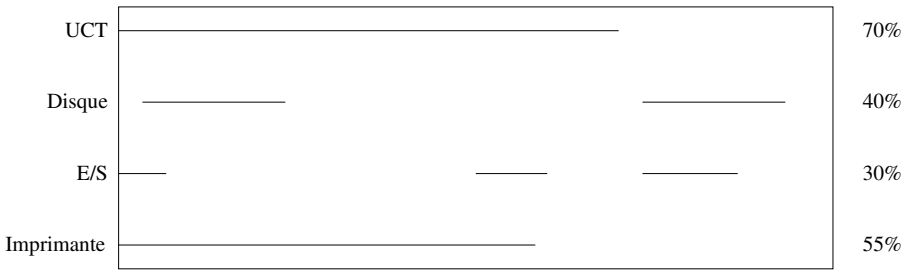

Figure 1.4 – Graphe de Gantt

Le graphe de Kiviat met l'accent sur les interrelations entre les composants. Dans un cercle, on dessine un rayon pour chaque composant. Sur chaque rayon, on place un point correspondant au total dùtilisation puis on relie chaque point. Plus la surface est grande, plus le système est optimal. La figure 1.5 illustre un graphe de Gantt et la graphe de Kiviat correspondant.

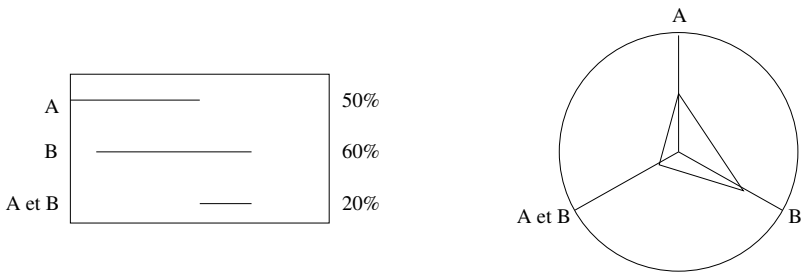

Figure 1.5 – Graphe de Kiviat

Le facteur d'étirement de la multiprogrammation

Cette mesure peut être utilisée pour évaluer l'impact de la multiprogrammation sur le temps de virement d'un programme. Elle est définie comme le ratio entre le temps de virement d'un programme dans un environnement multiprogrammé et le temps de virement du même programme lorsqu'il est exécuté sur un système dédié.

Le niveau de multiprogrammation

Il est défini comme le nombre de programmes en exécution simultanée qui compétitionnent pour le matériel et le logiciel.

Le taux de pagination

Cette mesure est utilisée sur les systèmes utilisant la mémoire virtuelle. Elle est définie comme la fréquence avec laquelle les programmes constituant la charge de travail génère des références à des éléments d'information qui ne sont pas présent en mémoire centrale.

Ces références (fautes de pages) causent des «page-in» et parfois des «page-out».

Le temps de réaction

Cette mesure représente le temps requis par le système pour réagir à une commande externe. C'est l'intervalle de temps entre la réception d'une commande (return ou soumission dans un traitement par lots) et le moment où l'UCT commence à l'exécuter (i.e. lui accorde une première tranche de temps).

Résumé

Nous avons donc vu une série de mesures de performance. Nous pouvons maintenant les classer.

Mesures externes	Mesures internes
Temps de virement	Utilisation
Temps de réponse	Chevauchement
Débit ou rendement	Facteur d'étirement de la multiprogrammation
Capacité	Niveau de multiprogrammation
Disponibilité	Taux de pagination
Fiabilité	Temps de réaction

1.4 Techniques d'évaluation

Nous possédons maintenant diverses mesures d'évaluation. Maintenant, nous abordons les différentes techniques disponibles pour effectuer des mesures. Ces techniques sont des méthodes avec lesquelles on obtient des valeurs pour chacune des mesures de performance.

On peut diviser les techniques d'évaluation en deux grandes catégories : les techniques empiriques et les techniques de modélisation.

1. Les techniques empiriques sont basées sur des mesures directes du système à évaluer. Le système doit donc exister et fonctionner. Pour effectuer les mesures, l'évaluateur doit posséder les outils nécessaires dont nous parlerons plus tard. Il est très important de noter que les mesures ne doivent pas être nécessairement faites lorsque le système traite une charge naturelle. Il est souvent possible et même plus commode d'utiliser une charge de travail de tests. [5]

2. Les techniques de modélisation sont de deux types : les techniques de simulation et les techniques analytiques.

5. Les différents types de charge de travail seront expliqués en détail plus tard.

1.4.1 La simulation

La simulation est une technique très valable pour évaluer la performance d'un système. Un modèle de simulation peut reproduire certains aspects du comportement d'un système même si ce système n'existe pas. On peut alors prendre des mesures réelles à l'aide d'outils de simulation et évaluer le comportement du système sous certaines circonstances.

On peut, jusqu'à un certain point, considérer la simulation comme une technique empirique. Toutefois, elle n'exige pas l'existence du système réel. En fait, toutes les techniques de modélisation offrent cet avantage.

La simulation, comme toutes les autres techniques de modélisation, offrent l'inconvénient d'utiliser des modèles qui sont inévitablement des représentations partielles et approximatives de la réalité. Ils fournissent donc des résultats dont la crédibilité est douteuse du moins jusqu'à ce que l'on ait de bons arguments sur la validité du modèle.

La simulation est tout de même un bon outil pour évaluer des systèmes inexistants. Évidemment le système réel devra être construit et testé pour prouver que la simulation est valide. Toutefois, la simulation permettra de prévenir la construction de système mal conçu en faisant ressortir leurs problèmes avant la construction.

La simulation par ordinateur est devenue très populaire dans l'industrie du transport et de l'espace. Cette popularité est particulièrement due aux conséquences sévères d'un échec.

Il existe deux types de simulateurs :

- Les simulateurs dirigés par les événements.

  Ce type de simulateurs est contrôlé par des événements qui se produisent pendant la simulation selon une distribution statistique spécifiée par l'usager.

- Les simulateurs dirigés par des scripts.

  Ce type de simulateur est contrôlé par des données empiriques (dérivées d'expériences) qui sont soigneusement manipulées pour réfléter le comportement anticipé du système simulé.

La simulation possède cependant certains inconvénients. La conception et l'implantation d'un simulateur sont des tâches complexes. La simulation elle-même requiert une certaine expertise de la part de l'évaluateur et beaucoup de temps ordinateur. Les simulateurs produisent généralement une certaine quantité de données qui doivent ensuite être analysées manuellement ou par ordinateur.

Cependant, une fois le simulateur développé, on peut l'utiliser de façon répétitive de façon utile et économique.

1.4.2 Les techniques analytiques

Les techniques analytiques diffèrent de la simulation par la nature des méthodes employées pour étudier les modèles. Ils utilisent des méthodes d'analyse mathématique plutôt que des techniques de simulation. Les modèles mathématiques sont donc des représentations mathématiques d'un système ou des composants d'un système.

Plusieurs types de modèles sont utilisés, les plus populaires étant la théorie des files d'attente (queuing theory) et les chaînes de Markov. Ce sont les plus utiles et les plus simples à utiliser. Pour les évaluateurs qui possèdent des connaissances mathématiques, les modèles analytiques peuvent être relativement facile à créer et à modifier.

Un grand éventail de résultats mathématiques peuvent être appliqués pour aider à estimer la performance d'un système informatique donné ou d'un composant assez rapidement et de façon relativement précise dans bien des cas.

Toutefois les modèles analytiques possèdent les inconvénients suivants :

- Les évaluateurs doivent avoir une certaine habileté en mathématique, ce qui n'est pas le cas dans les environnements de traitement de données commerciales.
- De belles solutions existent seulement pour des modèles simples. Pour les modèles plus complexes, l'évaluateur peut avoir certaines difficultés à trouver une solution mathématique précise qui décrit le comportement du modèle. Cela limite la liberté de l'évaluateur.
- Les systèmes d'aujourd'hui sont souvent si complexes que celui qui conçoit le modèle est forcé de faire plusieurs suppositions et simplifications. Ces suppositions et simplifications peuvent invalider l'utilité et l'applicabilité du modèle.
- L'évaluateur doit comprendre plusieurs techniques différentes et doit utiliser ces techniques en concert avec d'autres.

Les résultats d'une comparaison qui utilise une technique particulière peuvent parfois être invalidées par d'autres études qui utilisent d'autres techniques. Toutefois, il arrive très fréquemment que plusieurs évaluations différentes tendent à se renforcer l'une et l'autre. Cela aide à démontrer la validité des conclusions de l'évaluation.

1.5 La charge de travail

On appelle *charge de travail* d'un système, toutes les demandes de traitement, que ce soit de programmes, des données ou des commandes, faites par les usagers.

Chaque fois que la valeur d'une mesure de performance est donnée, la charge de travail avec laquelle cette valeur a été obtenue doit être indiquée. En effet, la performance ne peut être exprimée par des quantités indépendantes de la charge de travail. De plus, les comparaisons de performance entre les systèmes ou entre des configurations sont significatives seulement si la charge de travail traitée par les systèmes comparés est la même. Si ce n'est pas le cas, les différences entre les mesures refléterons les différences entre les charges aussi bien qu'entre les systèmes. Il sera très difficile ou même impossible de séparer les contributions de ces deux facteurs. Même une très petite différence entre les charges peut amener des perturbations.

Voici quelques exemples de perturbations qui peuvent affecter les résultats :

- la séquence des arrivées qui diffèrent (un programme A arrive avant B ou l'inverse) ;
- la présence ou non de programmes interactifs (dans un traitement par lots, l'exécution précédente n'affecte pas la suivante ; dans un système interactif, il y a influence).

La performance **P** d'un système **S** peut être évaluée seulement si les caractéristiques de la charge de travail **W** traitée sont explicitement prises en compte. Les relations entre **P**, **S** et **W** sont très complexes. En général, **W** varie continuellement dans le temps et cela est due à maintes causes. Ainsi, à l'intérieur d'un seul programme, le temps d'exécution peut varier selon ses données en entrée, l'allocation des ses fichiers de travail sur la mémoire secondaire, la position de la tête de lecture du disque lorsqu'une nouvelle demande arrive, etc. A ces causes de fluctuation qui peuvent

être considérées comme intrèsèques aux programmes dans **W**, on doit ajouter d'autres raisons externes telles l'organisation de l'installation et les caractéristiques des applications sur le système évalué [6]. Finalement, la charge est influencée par les algorithmes de contrôle du système (priorité dynamique ou non, allocation de la mémoire, ...).

1.5.1 Caractérisation de la charge de travail

La description quantitative de la charge de travail est appelée caractérisation de la charge de travail. La caractérisation est normalement faite en terme de paramètres de la charge qui peuvent affectés le comportement du système. Ceux-ci sont ensuite définis sous une forme utile dans le but de reproduire la charge.

On doit caractériser la charge de travail selon les buts de l'évaluation. Ainsi, l'analyse de ressources surchargées pendant de courtes périodes de pointe requiert une caractérisation de la charge de travail différente de la prédiction de l'impact sur la performance de l'ajout d'une nouvelle application.

Par étapes, on doit d'abord choisir les buts et caractériser la charge. Ensuite, on choisit la technique appropriée (modèle analytique, simulation ou système réel) pour prendre les mesures. On peut alors spécifier les paramètres nécessaires et la forme de leur représentation (valeurs simples, distributions, descripteurs de distributions, ...).

Voici quelques paramètres qui peuvent servir à caractériser la charge de travail :

1. pour les composants de base :
    - le temps UCT consommé par le programme ;
    - le nombre total d'opérations d'entrées/sorties du programme ;
    - l'espace mémoire demandée par le programme ;
    - la quantité de données lues par le programme ;
    - la quantité de résultats produits ;
    - la priorité externe du programme ;
    - le nombre de fichiers utilisés ;
    - le nombre d'unités différentes d'entrées/sorties utilisées (ruban, ...) ;
    - la fréquence d'exécution des différents types d'instructions ;
    - le temps moyen d'exécution entre deux entrées/sorties ;
    - le temps moyen d'exécution d'une entrée/sortie ;
    - la probabilité d'accès aux différents périphériques à chaque entrée/sortie ;
2. pour la charge totale :
    - le taux d'arrivée des travaux ;
    - la distribution de l'intervalle de temps entre chaque arrivée (ou chaque demande) ;
3. pour la charge interactive :

6. Par exemple, le temps de début et de fin de tests de gros logiciels, l'exécution périodique du système de paye ou de contrôle d'inventaire et toutes autres applications qui s'exécutent à des fréquences fixes.

- le nombre d'usagers ;
- le temps entre chaque commandes (où l'usager pense) ;
- la ratio entre le temps réponse et le temps de réflexion ;
- le ratio entre le temps moyen de service par interaction et le temps d'arrivée entre chaque commande ;

On appelle la charge de travail de tests la charge traitée pendant la prise de mesures. Dans certaines études d'évaluation, la charge de tests coincide avec la charge naturelle actuellement traitée par le système pendant la période considérée. Toutefois, ce type de charge est difficilement reproductible même si le même ensemble de programmes est soumis de nouveau au même système. La reproductibilité d'un charge est une condition essentielle pour une comparaison valide des résultats.

On appelle session de mesure l'intervalle de durée minimale pendant laquelle une expérience de mesure sera faite. Cette session peut-être subdivisée en sous-session. La durée minimale d'une session peut être établie après avoir spécifié la précision de la mesure.

1.5.2 Modèle et représentativité

La caractérisation de la charge de travail est fondamentalement importante dans tous les problèmes d'évaluation et est indispensable pour concevoir un modèle de la charge de travail.

On appelle modèle de la charge l'ensemble des programmes qui sont conçus et implantés pour charger le système artificiellement pendant la prise de mesures. Les plus importantes motivations pour utiliser des modèles de la charge réelle plutôt que cette dernière sont :

1. pour satisfaire le besoin de la reproductibilité de l'expérience. Cela rend les comparaisons entre les mesures utiles.
2. pour réduire substantiellement la durée de chaque session de mesure.
3. pour obtenir une représentation de la charge consistante avec son utilisation.
4. pour éviter les problèmes de sécurité et de confidentialité qui limitent quelquefois l'utilisation des programmes réels et des données réelles.

Une charge peut être modélisée à trois niveaux :

- **Physique**

  Le modèle est basé sur la consommation ou sur le taux de consommation des ressources matérielles ou logicielles du système. On utilise des éléments tels la consommation de l'UCT, la quantité d'entrées/sorties.

  Ce niveau est utile lors d'une évaluation dans le but de faire de la mise au point fine (fine tuning),de concevoir un nouveau système ou de faire de la planification de capacité (capacity planning).

- **Virtuel**

  À ce niveau, la consommation est basée sur la consommation de ressources virtuelles. Chaque composant de base peut donc être caractérisé par le nombre d'énoncés d'un langage évolué,

le nombre d'accès à chaque fichier ou base de donnée, le nombre ou le type de commandes interactives, la mémoire virtuelle utilisée, ...

Ce niveau est utile pour comparer des systèmes, pour la conception de nouveaux systèmes ainsi que pour la planification de capacité.

- **Fonctionnel**

  Ce niveau est caractérisé par les applications contenues dans la charge de travail. On utilise donc des fonctions tels le système d'inventaire, un programme de tri et une compilation.

  Ce niveau est utile pour comparer des systèmes, pour la conception de nouveaux systèmes ainsi que pour la planification de capacité.

La validité du modèle de la charge est une caractéristique essentielle pour la crédibilité et aussi l'utilité du modèle. On associe la validité d'un modèle à sa représentativité. La représentativité d'un modèle de la charge de travail est définie de différentes façons selon ces niveaux. Soit une charge C, certains critères pour évaluer un modèle C' de C peuvent être dérivés des définitions suivantes :

1. C' est un modèle parfaitement représentatif s'il demande les mêmes ressources physiques dans les mêmes proportions que C.
2. C' est un modèle parfaitement représentatif s'il demande les mêmes ressources physiques au même taux que C.
3. C' est un modèle parfaitement représentatif s'il traite les mêmes fonctions dans les mêmes proportions que C.

Il existe cependant d'autres facteurs à considérer :

- la relation des paramètres (des fonctions, de configuration, ...) avec la performance ;
- l'aspect dynamique de la charge ;
- le «compactness» de C'.

En résumé, il n'existe aucun critère absolu et unique pour construire et évaluer un modèle de la charge de travail. Le niveau de caractérisation et les paramètres caractérisés seront sélectionnés selon les objectifs de l'étude pour laquelle on fait la modélisation.

1.5.3 Charge de travail de tests

Nous avons déjà défini la charge de travail de tests comme la charge de travail traitée par un système pendant la prise de mesures. Elle peut être considérée comme un modèle de la charge de travail.

L'implantation du modèle de la charge de travail peut être faite grâce à diverses méthodes. Selon la méthode utilisée, les charges de travail de tests peuvent être classées dans les catégories suivantes :

- les charges de tests réelles ;

  Une charge de tests réelle est obtenue en prenant tous les programmes et données originaux traités pendant un intervalle donné.

- les charges de tests synthétiques ;

  Une charge de tests synthétique peut comprendre soit un sous-ensemble des composants de base (programmes, commandes interactives, ...) de la charge réelle ou un mélange de composants de la charge réelle et de composants construits pour cette étude (programmes synthétiques, programmes noyaux, scripts synthétiques, ...).

  On appelle le premier les charges naturelles synthétiques ou bancs d'essais (benchmark) et, le second, les charges synthétiques hybrides.

- les charges de tests artificielles ;

  Une charge de travail implantée sans utiliser aucun composant de la charge réelle est appelé une charge de travail artificielle. Ce type de charge peut se subdiviser en deux autres : les modèles artificiels exécutables et les modèles artificiels non exécutables. Ces derniers sont les entrées d'un système de simulation ou d'un modèle analytique.

La figure 1.6 illustre la hiérarchie des différents types de charges de travail.

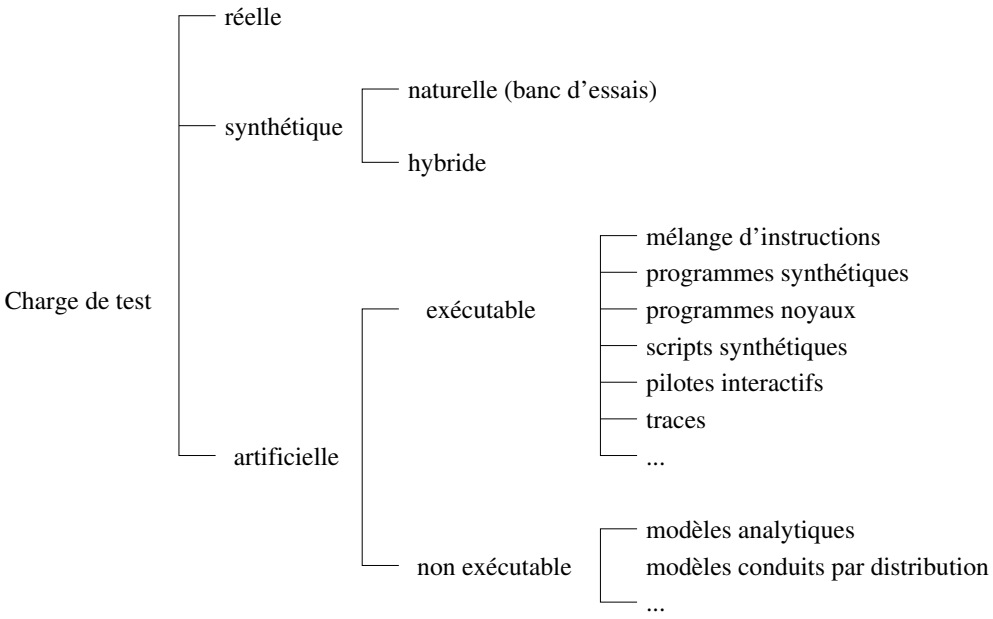

Figure 1.6 – Hiérarchie de charges de travail de tests

Charges de tests réelles

La charge naturelle d'un système peut être considérée comme un modèle. Ce type de charge est potentiellement le plus représentatif et sûrement le plus coûteux.

Avec un tel modèle, l'évaluateur n'a qu'un seul choix à faire, la durée de la mesure. En effet, il n'est pas nécessaire de mesurer toute la journée. On peut prendre des mesures pendant certaines

périodes cruciales. Le choix du moment propice et de la durée de la mesure est un problème complexe que nous n'aborderons pas ici.

Les raisons qui limitent l'utilisation de charges de tests réelles dans les expériences devant être répétées sous les mêmes conditions sont :

1. le manque de flexibilité dû à la non modifiabilité des programmes et de leur consommation de ressources ;
2. la nécessité de réutiliser les données originales (fichiers, ...) lorsque les programmes réels seront réexécutés. Toutes ces données devront être copiées sur la mémoire secondaire avec des pénalités économiques considérables et de l'interférence.
3. la confidentialité de certains programmes et de certaines données. Cela peut même amener l'interdiction de les copier et forcer leur remplacement par des programmes et des données ayant des caractéristiques de performance semblable.
4. les différentes organisations matérielles et logicielles de systèmes distincts ou les différentes versions du même système.

Charges de tests synthétiques

Un modèle comprenant des programmes extraient de la charge réelle à être modélisée est appelé charge synthétique naturelle ou banc d'essais (benchmark).

Les techniques utilisées pour implanter un banc d'essais sont les mêmes que celles utilisées pour construire n'importe quel autre modèle de charge exécutable.

Les difficultés rencontrées par l'utilisation d'un banc d'essais sont surtout dues à certaines conditions externes à l'ensemble des programmes. Ces conditions ont toutefois un impact remarquable sur la performance mesurée sur les différents systèmes. Voici quelques exemples de difficultés :

1. la priorité d'exécution ;

   Chaque système traite les priorités de différentes façons. Sur certains systèmes, la priorité assignée à un programme par le planificateur est altérée pendant l'exécution et cela affecte la séquence de l'exécution, le mélange des travaux et conséquemment la performance.

2. le degré maximum de multiprogrammation ;

   Ce paramètre varie de système en système et influence substentiellement les mesures de performance.

3. les routines d'écoute ;

   Ces routines sont souvent les sources des données utilisées dans les études d'évaluation. Chaque système possède ses propres routines qui parfois affectent des noms similaires à des variables ayant différentes significations. Avant de commencer une étude, il est raisonnable de connaître les significations précises des variables mesurées par les routines et les erreurs qui les affectent.

4. les paramètres de génération du système ;

   Chaque système d'exploitation possède plusieurs paramètres auxquels on assigne des valeurs à la phase de génération. Les valeurs de ces paramètres affectent la performance. C'est aussi le cas de d'autres paramètres qui peuvent être choisis par l'installation ou par les usagers.

Par exemple, les méthodes d'accès, les connexions canaux-périphériques, les options de compilation, la localisation en mémoire hiérarchique de modules du système d'exploitation et des fichiers, ...

Ces difficultés apparaissent lorsqu'un banc d'essais est exécuté par un vendeur sur sa machine. L'usager peut, pour y contrevenir, fournir au vendeur certaines règles à suivre lors de l'exécution du banc d'essais. Ces règles doivent adresser tous les points techniques et organisationnels qui peuvent invalidés les résultats de l'évaluation.

Les bancs d'essais possèdent plusieurs avantages. Faciles à concevoir, ils sont utiles dans un environnement complexe de multiprogrammation, de temps partagé, de multitraitement, de bases de données, de communication de données et de temps réels car ils s'exécutent sur une machine réelle dans des circonstances réelles. Les effets du logiciel peuvent être expérimentés plutôt qu'estimés. Les bancs d'essais sont aussi utiles pour évaluer le matériel aussi bien que le logiciel et cela même pour un environnement complexe. Ils sont aussi utiles pour comparer les opérations d'un système avant et après certains changements.

Les bancs d'essais ne sont toutefois pas tellement utiles pour prédire les effets de changements sauf si un système existe avec ces mêmes changements.

Les bancs d'essais sont très souvent utilisés pour établir un choix lors de l'acquisition d'équipements.

Vous trouverez en annexe plusieurs exemples de bancs d'essais communs.

Charges de tests synthétiques interactives

Lorsque l'on veut évaluer un système interactif, un modèle de charge interactive peut être implantée à partir d'un ensemble de scénarios. Un scénario est une description des fonctions exécutées par une série de commandes de l'usager. Ce scénario est spécifié dans un langage indépendant du système. Sa longueur doit être raisonnable car elle influence le coût du banc d'essais.

Afin d'être exécuté, un scénario doit être traduit en un script. Un script est une suite de commandes, exécutables sur un système particulier, qui sont envoyés au terminal et une suite de temps de réflexion qui s'insère entre chaque commande.

Une charge de tests synthétique naturelle (banc d'essais) pour un système interactif est composée de scripts extraits de la charge réelle.

Les outils Empower, Prevue et loadRunner construisent des scripts à partir de la charge réelle. Ils surveillent et enregistrent toutes les commandes interactives faites pendant une période donnée.

Programmes synthétiques et noyaux

Lorsqu'il est impossible pour diverses raisons (confidentialité, sécurité, ...) d'utiliser des programmes réels, ceux-ci sont remplacés par des programmes synthétiques et des programmes noyaux.

Un programme synthétique ne fait aucun travail utile, mais il consomme une certaine quantité de ressources du système qui sont fonctions des valeurs de ses paramètres de contrôle. Les valeurs de ces paramètres sont choisis par l'usager du programme. Ils déterminent des demandes de ressources tels le temps total d'UCT requis, le nombre d'opération d'entrées/sorties à effectuer, la quantité de mémoire requise, le nombre de lignes à lire ou à imprimer, le nombre de fichiers à accéder, etc.

Les programmes synthétiques peuvent avoir deux types de paramètres additionnels : les paramètres de correction et de calibration. Les paramètres de correction permettent de réduire les

influences qu'ont les paramètres les uns sur les autres. Ils sont utilisés pour éliminer ou réduire les distorsions non voulues de certaines consommations de ressources suite à des changements dans les paramètres de contrôle de d'autres ressources et qui, idéalement, ne devraient pas les influencer. Par exemple, si on change le nombre d'entrées/sorties demandé dans un programme, sa demande en temps UCT change. On utilisera donc les paramètres de correction pour réduire les distorsions et obtenir la valeur du temps UCT spécifié par le paramètre de contrôle.

Les paramètres de calibration sont utilisés pour la mise au point «fine» du programme synthétique pendant la phase de calibration. Cette phase est nécessaire car une combinaison des valeurs des paramètres de contrôle qui correspondent aux consommations de ressources désirées est rare. Après avoir déterminée la combinaison des valeurs d'entrées correspondant le plus près possible aux consommations de ressources, les paramètres de calibration sont utilisés pour améliorer la précision (ou validité) du programme synthétique.

Une des caractéristiques les plus importantes des programmes synthétiques est leur flexibilité. Elle leur permet de simuler un grand nombre de programmes réels du point de vue des consommations de ressources. Par l'intermédiaire de boucles imbriquées, on peut simuler les cycles UCT-entrées/sorties d'un programme réel en spécifiant le nombre d'instructions à exécuter par l'UCT et en contrôlant le nombre d'opérations d'entrées/sorties exécutées.

Des programmes synthétiques avec différentes structures peuvent être construits pour simuler différents types d'applications. Les programmes synthétiques peuvent aussi servir à évaluer les systèmes à mémoire virtuelle.

Il existe plusieurs programmes synthétiques commerciaux. [7]

Les programmes **noyaux** sont caractérisés par des consommations de ressources connues et non modifiables. Les programmes noyaux ont une application moins générale que les programmes synthétiques qui peuvent modéliser bien des programmes, mais ils sont disponibles plus rapidement car la phase de calibration n'existe pas pour eux.

Pour les **systèmes interactifs**, un programme **synthétique** est un script synthétique qui est très difficile à implanter. Parmis les paramètres possibles, on retrouve :

- le taux d'entrées des caractères (type-in) ;
- le temps de réflexion ;
- le nombre de fois qu'un certaine séquence de commandes est répétée dans le script.
- etc.

Un modèle de charge comprenant un mélange de programmes de la charge réelle et de programmes synthétiques ou noyaux est appelé hybride synthétique. Pour un système interactif, un modèle de charge hybride synthétique comprend des scripts de la charge réelle aussi bien que des scripts synthétiques.

Charges de tests artificielles

Un modèle artificiel d'une charge donnée comprend seulement des composants de base conçus pour charger artificiellement un système réel ou un modèle du système. Ainsi, on ne retrouve aucun composant de la charge à modéliser dans une charge artificielle. Cette classe de charges de tests comprend plusieurs catégories. Voici les principales charges de tests artificielles exécutables :

7. Je n'ai pas d'exemple pour l'instant. Je cherche....

1. **Vitesse de l'ordinateur** (timing)

   On peut évaluer certains ordinateurs en mesurant la vitesse de certaines instructions. On charge la machine avec une instruction particulière et on mesure la vitesse. On répète cette opération avec certaines autres instructions afin de pouvoir faire une comparaison.

   On appelle ce genre de mesure le «timing». On l'utilise seulement pour quelques opérations. On choisit donc les plus significatives. L'addition «add» est un exemple d'instructions significatives. On utilise le «timing» pour mesurer la vitesse des ordinateurs. Une unité fréquemment utilisée pour exprimer cette mesure est le MIPS (million d'instructions par seconde). On commence aussi à parler de BIPS (milliard d'instructions par seconde).

   Le «timing» est particulièrement utilisé pour comparer les ordinateurs d'une même famille. Par exemple, on peut s'en servir pour comparer entre eux les IBM, les LSI, les VAX, les SPARC, les PRIME, ...

   Le «timing» est une comparaison rapide et sa validité est limitée. Pour effectuer une validation plus significative des systèmes matériels et logiciels, on doit utiliser d'autres techniques. De plus, certains ordinateurs offrent un jeu d'instructions tellement grand que le «timing» devient peu significatif pour une application particulière.

2. **Mélange d'instructions**

   L'analyse de la fréquence d'exécution de différentes instructions peut fournir des informations utiles à la caractérisation de la charge. Tout programme ayant la même fréquence d'exécution d'instructions qu'un autre programme (ou d'un ensemble de programmes) peut être considéré comme un modèle précis du programme si l'on considère l'utilisation de l'UCT. Un modèle de ce type est un modèle par mélange d'instructions.

   Ce modèle peut comprendre un seul programme dont les fréquences d'exécution des instructions coincident avec celles de la charge à modéliser. Cette technique est en fait une moyenne pondérée du «timing» de plusieurs instructions significatives pour une charge particulière. Cette mesure est plus significative que le «timing» car un système peut faire des opérations en point flottant double précision très rapidement, mais il peut être très mauvais dans un environnement de traitement de données commerciales où le mouvement des données, l'édition, les entrées/sorties sont importantes. Le mélange d'instructions permet de mieux cibler les instructions utiles dans un environnement donné.

   Le mélange d'instructions était parmi les premières charges de tests utilisées pour évaluer la puissance de traitement de l'UCT. Si on tente de comparer plusieurs UCTs différentes, le mélange d'instructions doit être aussi indépendant de la machine que possible. On peut, avec le mélange d'instructions, comparer les machines avec plus de validité qu'avec le «timing».

   Cette méthode peut servir pour une comparaison rapide. Cependant, l'évaluateur doit utiliser d'autres approches pour effectuer une comparaison plus complète. De plus, il n'est pas toujours possible de dériver directement un mélange d'instructions à partir de la charge actuelle. C'est le cas d'un système déjà conçu mais non installé sans aucun prédécesseur à remplacer. Dans ce cas, on ne peut dériver un mélange d'instructions. Il faut donc utiliser un mélange d'instructions d'une autre installation supportant une charge similaire ou utiliser des mélanges commerciaux appelés «application mixes».

   Le concept de mélange d'instructions peut aisément s'étendre du domaine de mélange d'instructions à celui de mélange d'énoncés de langage évolué. On appelle ce dernier, un mélange

d'énoncés.

Parmi les facteurs qui influencent la puissance de traitement de l'UCT, on retrouve :

- des facteurs reliés à la mémoire (dimension de la mémoire tampon (cache), politique de gestion, ...) ;
- des facteurs reliés à l'ordre d'exécution des instructions (pré-chargement des instructions et des opérandes, tampons d'instructions, vitesse de l'horloge, ...) ;
- des facteurs reliés à la manipulation d'adresses (calcul d'adresse effective, traduction de l'adresse virtuelle à physique, ...).

Dans un mélange d'instructions, toute l'information reliée à l'ordre d'exécution des instructions est perdue. Les mélanges d'instructions sont donc inadéquats pour évaluer la performance d'un UCT influencé par l'ordre d'exécution des instructions. La validité de cette technique devient de plus en plus difficile à établir avec l'arrivée de matériels de plus en plus complexes. Avec l'utilisation de mémoire tampon (cache), du pipeline et d'unités multiples (super-scalaire) la même instruction peut s'exécuter plus ou moins rapidement selon le contexte de ses différentes exécutions. Cette méthode peut cependant permettre d'évaluer la performance optimale (peak performance) d'une machine.

Le mélange d'instructions est basé sur une répartition dynamique. Cela signifie que le «nombre» d'instructions varie selon les instructions et les applications. Un façon d'obtenir ces fréquences dynamiquement consiste à ajouter des instructions de comptage aux programmes à modéliser. Après l'exécution du programme modifié, les fréquences des différentes instructions exécutées sont retrouvées en mémoire à des endroits réservés à cet usage. Ces fréquences sont souvent différentes de celles qui peuvent être dérivées des «listings» des programmes. Cette dernière mesure serait une répartition statique. Un mélange d'instructions statique peut s'utiliser, par exemple, pour la conception de compilateur.

Un dernier défaut des mélanges d'instructions est qu'il ne permet pas ou presque d'évaluer le logiciel.

3. **Programmes synthétiques ou noyaux**

   Les autres modèles de charge artificielle exécutable sont obtenus en regroupant des programmes synthétiques et noyaux. Contrairement au mélange d'instructions, ce type de charge est un regroupement de programmes et non d'instructions.

   Ces deux types de programmes ont été définis à la section précédente. Nous ne reviendrons pas sur ces définitions.

   Lorsque ces types de programmes sont utilisés, des problèmes de conception, d'implantation et de calibration surgissent.

   Les mêmes considérations s'appliquent aux scripts synthétiques ou noyaux pour les systèmes interactifs.

   Il existe deux solutions pour exécuter un script synthétique. Ainsi, pour simuler l'environnement d'un usager interactif, on a souvent recours à des programmes qui génèrent des commandes ou des messages pour simuler un usager. Ces programmes interceptent les messages dirigés vers chaque usager et soumettent le prochain message (ou commande) prescrit par le script qu'ils suivent. Avant de soumettre une nouvelle commande, ils attendent un certain

temps équivalent pour la sortie à l'écran, la réflexion et le temps pour l'entrée au clavier. Ce temps est spécifié par le script.

Si ces programmes s'exécutent sur le système mesuré, ils sont appelés des **pilotes internes**. S'ils sont connectés au système mesuré, ils sont appelés des **pilotes externes**. La présence de ces derniers n'influence pas de façon appréciable la charge du système mesuré. En revanche, les pilotes internes perturbent le système mesuré en produisant une charge inutile qui est l'exécution du programme contenant le script. De plus, les pilotes internes apportent un problème de représentativité venant du fait qu'ils éliminent toutes les opérations d'entrées/sorties au terminal. Ils sont cependant moins coûteux à produire que les pilotes externes.

Les deux types de pilotes ont été employés dans un grand nombre d'études. Voici quelques années une telle étude a été faite au département en utilisant des pilotes externes. L'expérience consitait à mesurer la performance d'une version légère de unix (Venix), qui s'exécutait sur des LSI 11/23 de la compagnie Digital. Un LSI était évalué et un autre exécutait les scripts interactifs. Les deux machines étaient reliées par quatre lignes séries. Le simulateur exécutait des scripts qui simulaient des usagers et transmettaient les commandes au système simulé par l'intermédiaire des lignes séries. Cette approche permettait de simuler quatres usagers (scripts) travaillant sur quatres terminaux (lignes séries).

Certains pilotes avancés peuvent construire le script et l'exécuter. Ce sont des pilotes externes qui capturent les commandes des usagers, en font un script et l'exécutent lors d'une éventuelle scéance de mesures. Ces outils sont de deux types :

- les outils basés sur un journal.
  Ces outils sont des moins intelligents. Ils exécutent le script sans s'adapter aux réponses ni aux divers événements qui peuvent survenir.
- les outils basés sur les émulateurs.
  Ces outils exécutent le script de façon intelligente. Il reconnaît les réponses, mesure les temps réponse et décide quand soumettre la prochaine entrée. Ce type d'outil est cependant plus coûteux.

Ces pilotes peuvent fonctionner en mode ASCII ou graphique. Voici des exemples de tels pilotes :

- Empower et Empower/X de Performix ;
- Prevue et Prevue/X de Performance Awareness Corp ;
- Load Runner et XRunner de Mercury.

L'alternative à ces outils consiste à utiliser des usagers humains suivant un script synthétique. Cela n'est pas très pratique étant donné son coût élevé en terme de ressources humaines ou techniques et à la reproductibilité limitée de l'expérience.

4. **Autres**

   Le coût de conception et d'implantation de modèles complexes de la charge peut être très élevé. Dans quelques études, une mesure simple concernant l'importance ou la magnitude de la charge peut suffir. Cette étude compare la magnitude de la charge courante avec celle d'une charge artificielle composée de n copies d'une commande ou d'un script. On appelle cette charge artificielle une sonde (terminal probe).

Une sonde est envoyée d'un terminal à chaque fois que la charge d'un système doit être mesurée. Comme la sonde est toujours la même, les différences dans le temps réponse obtenues dans différentes utilisations de la sonde seront attribuées aux différences dans le mélange de programmes et de commandes.

Pour exprimer la magnitude de la charge, le temps réponse de la sonde est aussi mesuré sur un système dédié lorsqu'il exécute 1, 2, 3, ..., n copies de la sonde. Une charge est équivalente à x sondes si le temps réponse de la sonde coïncide à celui obtenu lors de l'exécution de x copies de la sonde sur le système dédié.

1.6 Principe de mesure

Pour étudier un système, on doit utiliser diverses techniques de mesure. Mais que signifie *mesurer un système* ?

Généralement, cela signifie amasser de l'information sur les activités d'un système pendant qu'il sert des usagers, qui peuvent être réels ou simulés. Dans certains cas, l'information est amassée en observant un modèle du système (un simulateur) qui peut prédire le comportement du système réel lorsqu'il servira les usagers.

Dans cette section, nous expliquerons comment mesurer un système en présentant les différents principes de base de la prise de mesure et les différentes techniques disponibles.

Les mesures peuvent être classifiées en deux catégories :

1. les mesures demandées par les usagers du système ;
2. les mesures demandées par le système.

Les mesures concernant l'utilisation des ressources du système faites dans le but d'évaluer le système, de contrôler son utilisation et de planifier l'ajout de nouvelles ressources, appartiennent à la première catégorie.

Les mesures prises par le système pour se gouverner lui-même et qui lui permettent de s'adapter dynamiquement aux facteurs conditionnant son activité, appartiennent à la seconde catégorie. Ces mesures peuvent permettre au système d'ajuster dynamiquement certains paramètres ou de modifier l'arrivée des demandes aux ressources. Un exemple d'ajustement dynamique est la revision périodique des priorités selon les valeurs mesurées de l'utilisation de l'UCT et autres ressources. La modification dynamique des taux d'arrivée se fait au moyen de boucles rétroactives. Une boucle rétroactive (feedback loop) est une situation dans laquelle l'information sur l'état courant du système est mise disponible à toutes les demandes qui arrivent. Ces demandes peuvent alors être détournées si la rétroaction indique qu'ils peuvent avoir des difficultés à être servies. La rétroaction peut être négative ou positive.

- Rétroaction négative ;

  La rétroaction est négative si le nombre d'arrivées des nouvelles demandes peut décroître suite à l'information retournée par la rétroaction. Ainsi, un automobiliste arrivant à une station service et observant qu'il y a plusieurs voitures en attente à chaque pompe, ira à une autre station moins occupée.

  Dans les systèmes d'exploitation, on peut appliquer cela à toutes les files d'attente telle la file de travaux à imprimer.

Cette rétroaction contribue à la stabilité des systèmes de files d'attente. Si des travaux entrent sans discrimination dans la file d'une ressource occupée, cette file peut s'accroître indéfiniment. La rétroaction négative aide à conserver une valeur moyenne dans la file.

- Rétroaction positive.

  La rétroaction est positive si le nombre d'arrivées des nouvelles demandes peut augmenter suite à l'information retournée par la rétroaction. Un problème avec la rétroaction positive est l'instabilité qui peut en résulter. L'exemple de l'écroulement d'un système paginé illustre bien ce cas. Dans un système particulier, si l'UCT est sous-utilisée, on va tenter d'augmenter son utilisation en admettant de nouveaux processus. Cependant, la sous-utilisation de l'UCT peut être causée par un manque de mémoire entraînant des attentes continuelles pour la fin d'entrées/sorties sur des fautes de pages. Comme d'autres processus arrivent, la quantité de mémoire allouée à chacun diminue et le nombre de fautes de pages augmente. Le résultat est une diminution de l'utilisation de l'UCT. Si le système est mal conçu, il tentera à nouveau d'augmenter l'utilisation de l'UCT en admettant de nouveaux travaux. En répétant ces étapes, on obtiendra un écroulement du système.

  Les concepteurs de systèmes utilisant la rétroaction positive doivent être très soigneux lors de la conception de tels mécanismes pour prévenir la possibilité du développement de comportement instable. On peut, dans ces cas, étudier les effets de chaque changement afin de vérifier si l'on obtient les résultats anticipés. Si cela n'est pas le cas, cela peut indiquer un état instable et le système d'exploitation doit s'ajuster (modifier l'allocation des ces ressources jusqu'à ce que la stabilité soit revenue).

En général, aucun système n'est conçu pour être mesuré. Cela restreint les possibilités de mesures du système et requiert la modification de certaines de ses fonctions ou l'ajout d'instruments coûteux si on veut enlever ces restrictions.

La mesurabilité d'un système informatique est définie comme une fonction de l'information que l'on peut obtenir avec un moniteur par rapport au coût des mesures. Ainsi la mesurabilité peut varier entre deux extrêmes, correspondant à la possibilité de mesurer chaque composant du système individuellement au niveau de détail désiré et à l'inaccessibilité totale au système. Dans ce dernier cas, seulement le temps réponse à différentes situations peut être mesuré.

Pour rencontrer les objectifs d'une étude d'évaluation, il est parfois nécessaire de déterminer les mesures globales tels le temps de virement moyen, le temps de réponse moyen et l'utilisation des unités périphériques. Dans ce cas, on a une analyse macroscopique.

Lorsqu'un haut niveau de détails est nécessaire, on utilise des mesures tels la contribution de chaque type d'instructions à l'utilisation de l'UCT, le nombre de pages chargées pendant un certain intervalle de temps, ... Dans ce cas on fait une analyse microscopique. Le facteur distinctif entre ces deux types d'analyse est la durée du phénomène observé et la fréquence à laquelle ils se produisent.

De manière générale,nous nommerons moniteur tout le logiciel ou matériel servant à mesurer un système.

Techniques de mesure

Il existe deux grandes techniques pour mesurer un système informatique. Le choix dépend du type d'analyse désiré et du niveau auquel on effectue l'analyse. Les deux grandes techniques sont

par détection d'événements ou par échantillonnage.

Mesure par détection d'événements

Un événement est un changement dans l'état du système. Une technique pour collecter les données sur certaines activités du système consiste donc à capturer tous les événements associés à cette activité et de les enregistrer dans l'ordre d'arrivée. Dans ce cas la mesure est faite par détection d'événements.

Le début d'une opération d'entrées/sorties, le passage de l'UCT de l'état occupé à inactif et les interruptions sont des exemples d'événements.

On peut classer les événements en deux types :

- les événements logiques ;

  Ces événements se produisent losrqu'un programme atteint une certaine étape de son exécution. Par exemple, lorsqu'il débute une entrée/sortie (appel système).

- Les événements matériels.

  Ces événements correspondent à l'apparition de signaux dans les circuits d'un composant. Le mouvement de la tête d'un disque en est un exemple.

Plusieurs événements matériels peuvent être reconnus par le logiciel car ils modifient certaines locations en mémoire. Ainsi, lorsqu'on parle de mesure par détection d'événements, on parle des événements reconnus par le logiciel et des événements logiques.

La détection des événements se fait par l'insertion de bouts de codes spéciaux à des endroits spécifiques du système d'exploitation. Lorsqu'un événement à intercepter se produit, ce code transférera le contrôle à la routine appropriée qui enregistre l'événement et toutes les informations utiles. L'ensemble des instructions et des données utilisé à cet effet constitue un **moniteur**.

Un moniteur est donc un mécanisme qui collectionne l'information sur les activités du système. Ce type de moniteur est dit *conduit par les événements* (event-driven). Il collecte une **trace** de tous les événements désirés. Les données emmagasinées par le moniteur seront ensuite analysées. Elles seront d'abord réduites pour les rendre plus facilement interprétable.

L'utilisation de la technique de mesure par détection d'événements doit être sélective. En effet, si on intercepte trop d'événements, les opérations normales du système seraient grandement ralenties. Plus on intercepte d'événements, plus le moniteur cause de «l'overhead» et perturbe le fonctionnement normal du système. Un autre facteur important à considérer est l'utilisation de l'espace pour emmagasiner les données. On doit pour cela utiliser un tampon en mémoire centrale pour ne pas écrire trop souvent sur la mémoire secondaire. Le tampon doit être assez grand pour éviter, lorsqu'un événement d'entrées/sorties se produit, d'attendre la fin de l'entrée/sortie pour le tampon. Si cela se produit, on risque de perdre des événements. On utilise donc fréquemment deux tampons. Pendant qu'un tampon est enregistré sur la mémoire secondaire, les événements sont emmagasinés dans le second.

Il est facile d'imaginer comment on peut implanter un tel moniteur lorsqu'il s'agit de détecter des événements du type interruption. On remplace alors les routines d'interruption par une routine de mesure. Lorsqu'il s'agit de détecter des événements logiques, le moniteur est plus difficile à implanter. Cela implique des modifications au système d'exploitation et c'est très coûteux en terme de ressources nécessaires.

La technique de mesure par détection d'événements possède l'avantage de pouvoir fournir plus d'information que les autres techniques. Cependant, elle ne peut être utilisée si les outils ne sont pas inclus dans le système d'exploitation. Il est pratiquement impossible d'inclure de tels outils dans le système.

Mesure par échantillonnage

Une technique souvent préférée à la précédente est la prise de mesures par échantillonnage. Cette technique consiste à interrompre le système à intervalle régulier afin de détecter l'état de certains de ses composants. Si le nombre d'échantillons est assez grand, ce type de mesure est très valable.

La méthode par échantillonnnage est en fait une technique statistique. Elle est utilisable même si le mesurage de toutes les données concernant un ensemble de personnes, d'objets ou d'événements est impossible ou trop coûteux. Au lieu d'examiner tout l'ensemble (ou toute la population), la méthode examine seulement une partie appelé un **échantillon**. De cet échantillon, il est possible d'estimer, souvent avec une grande précision, certains paramètres qui caractérisent la population.

Lorsqu'elle est utilisée pour mesurer la performance d'un système informatique, cette technique présente l'avantage de produire beaucoup moins de données que la technique par événements. Cela réduit et simplifie l'analyse.

La collecte des données par échantillonnage cause moins de dérangements dans le système que la détection d'événements par logiciel. On considère donc dans ce cas que l'effet de l'outil sur la performance du système est négligeable et facilement contrôlable.

La difficulté de cette approche consiste à choisir un bon échantillon.

1.7 Représentation des données

Lorsque les données sont amassées, il existe plusieurs façons de les représenter afin de faciliter leur interprétation. On peut les garder sous forme numérique ou bien les transformer sous forme graphique. Des exemples de représentations graphiques sont :

- les tables ;
- les diagrammes (graphes de Kiviat, disgramme de gantt, ...) ;
- les histogrammes ;

1.8 Instrumentation

L'ensemble des outils utilisés pour une étude est appelé instrumentation. Un instrument de mesure sert à quantifier les résultats d'une observation. Il existe deux types d'outils :

- les moniteurs matériels ;

  Ce sont des instruments spéciaux ou des ordinateurs spécialisés qui sont utilisés pour prendre les mesures. Les moniteurs matériels ont l'avantage de consommer peu de ressources et, conséquemment, ils possèdent l'avantage de pouvoir fournir des données plus précises. Cependant, certains types de données ne peuvent être fournies par un moniteur matériel. Finalement, ils sont souvent assez coûteux.

Un exemple de moniteur matériel est un ordinateur de diagnostic pour le réseau. C'est un ordinateur contenant un logiciel spécialisé qui se connecte sur un lien Ethernet et écoute toutes les communications. Il peut classer les messages, rapporter les erreurs, ...

- les moniteurs logiciels.

  Ces moniteurs sont des programmes ajoutés aux systèmes à mesurer. Ils ajoutent à la charge du système, mais cet effet peut être considérablement réduit. Il y a cependant certains types de données que ces moniteurs sont incapables de fournir.

1.9 Étude de cas

1.9.1 Sun/Solaris

Le noyau de Solaris contient plusieurs compteurs qui sont utilisés pour garder une trace de plusieurs activités du système. Les activités et entités du système qui sont considérées sont :

- l'utilisation de l'UCT ;
- l'utilisation des tampons ;
- les entrées/sorties sur disques et rubans magnétiques ;
- les entrées/sorties sur les terminaux ;
- les appels système ;
- les changements de contexte ;
- les accès aux fichiers ;
- les différentes files d'attente ;
- les tables du noyau ;
- les communications inter-processus ;
- la pagination ;
- l'allocation de l'espace en mémoire centrale et de l'espace de «swap» ;
- l'allocation de la mémoire «noyau».

Pour accéder à ces informations, Solaris fournit des librairies et des outils interactifs.

Les librairies de Solaris (au nombre de deux) permettent à un usager de programmer ses propres applications. Ces librairies sont : kstat et kvm. kvm est un API pour accéder aux valeurs des différents compteurs du noyau. Les fonctions de kvm sont : kvm_open, kvm_nlist, kvm_read, kvm_write et kvm_close. Pour utiliser ces fonctions, on doit avoir les permissions du «super-user». Pour remédier à ce problème, Solaris fournit une autre librairie qui permet d'accéder à ces informations : kstat. Dans cette section, nous parlerons seulement des outils interactifs et non pas des librairies.

Solaris fournit plusieurs outils interactifs. Ce sont :

- les utilitaires *sar* et *sadc* ;
- la commande *ps* ;
- les utilitaires *perfmeter* ;
- les commandes *vmstat* et *iostat* ;
- la commande *swap* ;
- les commandes *netstat* et *nfsstat* ;

Il existe aussi certains outils qui ne sont pas intégrés dans Solaris :

- l'utilitaire *tops* ;
- les outils basés sur *se* ;

Identification de la charge de travail

Certaines commandes générales permettent de se faire une première idée de la charge du système. Les commandes *w* et *ps* offrent un premier survol.

Pour bien connaître la charge d'un système, on peut utiliser le système de comptabilisation (accounting) de Unix. Avec ce système, on peut identifier la fréquence d'exécution des programmes ainsi que l' utilisation en UCT, en mémoire et en unités d'entrées/sorties de chacun de ces programmes. Le système de comptabilisation de Solaris ajoute une charge considérée négligeable. Il y a cependant un script qui permet de traiter les données accumulées qui consomment plus de ressources. Il n'est cependant pas exécuté fréquemment.

Outils pour la prise de mesures

De même que pour identifier la charge de travail, il existe des commandes générales qui permettent de prendre certaines mesures. Les commandes *time* et l'outil *perfmeter* en sont des exemples.

Il existe aussi un ensemble d'outils beaucoup plus précis qui permettent de mesurer la plupart des opérations de Solaris. Ces outils sont *sar*, *iostat*, *vmstat*, *nfsstat* et *netstat*.

Outils pour mesurer les entrées/sorties

Pour contrôler la charge des unités d'entrées/sorties, on peut utiliser la commande *iostat*. Cette commande permet de :

- voir le nombre de caractères lus ou écrits au terminal par seconde (tin,tout) ainsi que de l'information sur la charge de l'UCT (**us**er, **sy**stem, **w**ai**t** et **id**le). La colonne *wait* indique le temps moyen d'attente dans la liste des processus prêts. Les options *-t* et -c permettent de voir ces valeurs.

```
>iostat -tc

       tty         cpu
```

```
 tin tout us sy wt id
 172 2172 29 22  5 44
```

- voir, avec l'option *-d* le débit de chaque disque (Kbps), le temps de service moyen pour chaque commande (serv) ainsi que le taux de transferts par seconde (tps).

  Voici un exemple de sortie sur carouge :

```
>  iostat  -d

          fd0           sd1           sd3           sd6
 Kps tps serv  Kps tps serv  Kps tps serv  Kps tps serv
   0   0  730    2   0   53   14   2  193    0   0   84
```

  La commande iostat prend par défaut les options -tcd.

- voir, en utilisant l'option *-D*, le nombre de lectures (rps) et d'écritures (wps) par seconde sur chaque disque ainsi que le % d'utilisation du disque (util). Voici un exemple de sortie avec cette option :

```
>  iostat  -D

          fd0           sd1           sd3           sd6
 rps wps util  rps wps util  rps wps util  rps wps util
   0   0  0.0    0   0  0.3    0   1  2.2    0   0  0.0
```

- voir des statistiques détaillées sur chaque disque. Cela est possible avec l'option *-x*. Parmi les données disponibles, le nombre moyen de commandes en attente (wait), le nombre moyen de commandes en traitement (actv), le temps de service (svc_t), le pourcentage de temps que le disque est occupé (%b) et le pourcentage du temps où il y a des demandes en attente (%w). Les autres données sont des versions plus détaillées des données des autres options (divisées en lecture et écriture).

  Voici un exemple de sortie :

```
>  iostat  -x

                                 extended disk statistics
disk      r/s  w/s   Kr/s   Kw/s wait actv  svc_t  %w  %b
fd0       0.0  0.0    0.0    0.0  0.0  0.0  730.5   0   0
sd1       0.2  0.0    0.7    1.1  0.0  0.0   53.0   0   0
sd3       0.4  1.5    2.9   11.2  0.0  0.4  192.7   0   2
sd6       0.0  0.0    0.0    0.0  0.0  0.0   83.7   0   0
sd15      0.9  1.1    5.0    8.4  0.0  0.1   46.1   0   2
```

```
sd16      0.9  1.1    5.0    8.4  0.0  0.1   44.5   0   2
sd17      0.7  0.8    4.4    6.9  0.0  0.2  163.4   0   1
sd18      0.3  0.2    2.5    3.2  0.0  0.0   31.8   0   1
sd31      0.5  0.2    2.3    1.2  0.0  0.0   91.3   0   1
sd33      0.8  5.5   10.7   34.6  0.0  0.3   47.6   0   7
sd36      0.0  0.0    0.1    0.0  0.0  0.0  137.9   0   0
```

Pour plus d'information, vous pouvez consulter le manuel interactif de Solaris concernant la commande iostat (man iostat)

La commande *df*, qui affiche de l'information sur le système de fichiers, est un autre outil disponible.

Les spécialistes de SUN ont établis quelques conclusions suite à différentes évaluations [8] :

- Un temps de services supérieur à 50ms est considéré comme trop lent.
- Si le taux d'utilisation d'un disque est supérieur en moyenne à 20% sur une période de plus de 30 secondes, on devrait songer sérieusement à vérifier son temps de service.
- le nombre de disque que peut supporter un contrôleur est :
  - trois disques si le contôleur est de type IPI ;
  - trois ou quatre disques actifs si le contôleur est de type fast SCSI (10Mb/s) ;
  - six ou huit disques actifs si le contôleur est de type wide SCSI (20Mb/s) ;

  Un nombre plus élvé de disques peut entraîner un temps d'attente possiblement trop long.

Outils pour mesurer NFS

Dans Solaris, le système de fichier NFS occupe une place importante et on doit donc aussi contrôler sa performance. On fait donc aussi de la mise au point sur NFS.

L'outil de mesure de NFS est *nfsstat*. Cet outil permet d'obtenir des statistiques sur les serveurs NFS (*-s*), sur les clients (*-c*) ainsi que sur les RPC (*-r*).

L'information retournée concerne le nombre total d'appels, le nombre d'appels refusés ou fantômes, le nombre de commandes de chaque type, les «timeout», etc. Voici quelques exemples de sorties NFS :

```
>nfsstat -s

Server rpc:
Connection oriented:
calls      badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
13373550   0          0          0          0          643909     16
```

8. La plupart des conclusions sont tirées d'une évaluation faite pour un serveur NFS en utilisant le banc d'essais LADDIS

```
Connectionless:
calls      badcalls   nullrecv   badlen     xdrcall    dupchecks  dupreqs
3339710    544        0          0          544        993        0

Server nfs:
calls      badcalls
16692473   14
Version 2: (3337867 calls)
null       getattr    setattr    root       lookup     readlink   read
0 0%       484721 14% 100 0%     0 0%       1087499 32% 1650 0%   1546831 46%
wrcache    write      create     remove     rename     link       symlink
0 0%       821 0%     63 0%      6 0%       0 0%       0 0%       0 0%
mkdir      rmdir      readdir    statfs
2 0%       1 0%       157738 4%  58435 1%
Version 3: (13289953 calls)
null       getattr    setattr    lookup     access     readlink   read
2537 0%    6944098 52% 114574 0% 3900934 29% 1386539 10% 3553 0%  238009 1%
write      create     mkdir      symlink    mknod      remove     rmdir
332858 2%  60123 0%   5644 0%    952 0%     0 0%       66124 0%   1395 0%
rename     link       readdir    readdir+   fsstat     fsinfo     pathconf
10376 0%   13543 0%   99756 0%   38316 0%   52324 0%   2113 0%    1028 0%
commit
15157 0%

Server nfs_acl:
Version 2: (0 calls)
null       getacl     setacl     getattr    access
0 0%       0 0%       0 0%       0 0%       0 0%
Version 3: (64753 calls)
null       getacl     setacl
0 0%       64753 100% 0 0%
```

Lorsque l'on veut évaluer la performance de NFS, on doit générer un mélange d'opérations qui soit significatif. La commande *nfsstat* nous fournit cette information. Le banc d'essais LADDIS utilise un mélange qui comprend 22% de lectures, 15% d'écritures, 34% de «lookup», etc.

Voici quelques conclusions sur la performance de NFS :

- on peut augmenter la performance de NFS en ajoutant des démons. Ceux-ci accélèrent le service et affectent peu la performance s'ils ne sont pas utilisés.
- les normes suivantes sont communes pour le nombre de démons NFS :
  - 2 thread/clients ;
  - 16 threads/ethernet ;
  - 160 threads/FDDI ;
  - 32 threads sur un serveur SPARC classic ;

– 64 threads sur un serveur SuperSPARC ;

Outils pour mesurer le réseau

L'outil *netstat* permet d'obtenir de l'information sur TCP/IP ainsi que de l'information sur les interfaces du réseau. Sur TCP/IP, il est possible d'avoir des informations sur les socket, les groupes, les STREAM, les tables de routages, etc.

Pour les interfaces, l'outil nous donne des informations sur le type de réseau, son adresse, le nombre de paquets reçus et envoyés, le nombre de paquets en erreurs, le nombre de collisions, etc.

Outils pour mesurer les processeurs

Solaris fournit quelques outils pour mesurer la charge des processeurs. Les instructions suivantes permettent d'obtenir des statistiques simples sur l'UCT :

- w – donne des informations sur les usagers.
- perfmeter – outil graphique pour obtenir des informations sommaires sur l'utilisation des différentes ressources du système.
- psrinfo – donne des informations sur les processeurs en utilisation ;
- psradm – permet de contrôler les processeurs ;
- mpstat – donne des statistiques détaillées sur les processeurs.

  Les informations fournies concernent le numéro de l'UCT, le nombre d'interruptions (intr), le nombre de changements de contextes (csw, icsw), le nombre de migrations de threads d'un processeur à l'autre (migr), le nombre d'appels systèmes (syscl), le pourcentage de temps UCT dans les différents modes (usr, sys,idl), etc. Voici un exemple de sorties pour *mpstat* :

```
> mpstat

CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0  125   1  190   382  256  437  115   42   35    0  1140   30  23   5  43
  2  125   1  190   382  256  453  116   42   35    0   356   29  22   5  44
```

- uptime – donne des informations sur le temps de fonctionnement de la machine ainsi que sur la charge moyenne du système.

```
> uptime
 3:49pm  up 8 day(s),  8:43,  13 users,  load average: 7.43, 7.57, 7.66
```

Outils pour mesurer la mémoire

Le principal outil pour mesurer la mémoire est *mpstat*. Cet outil fournit aussi beaucoup d'information sur toutes les ressources. Il peut fournir des informations générales sur :

- la mémoire utilisée et disponible, sur la pagination et le swapping. Voici un exemple de sortie :

```
> vmstat
 procs     memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr f0 s1 s3 s6   in   sy   cs us sy id
 2 0 0  13392 11444   3 251 29  8 18  0  6  0  0  2  0  665 1505  891 29 22 48
```

  La colonne *procs* affiche le nombre moyen de processus à l'état prêt, bloqué ou «swapped-out». La colonne «memory» indique la quantité d'espace disponible pour le swapping ainsi que la quantité de mémoire centrale disponible. La colonne «page» indique, en particulier le nombre de page-in et de page-out en kilo-octets. La colonne disque rapporte le nombre d'opérations par seconde sur le disque. La colonne «faults» indique le nombre de «trappes» (interruptions) par seconde (in = interruptions, sy = appels au système et cs = changements de contexte).

- la mémoire cache (*-c*). Voici un exemple de sortie :

```
> vmstat -c

flush statistics: (totals)
     usr     ctx     rgn     seg     pag     par
       0       0       0       0   63015       0
```

- le nombre d'interruptions par périphériques (*-i*). Voici un exemple de sortie :

```
> vmstat -i

interrupt         total     rate
--------------------------------
clock          72528605      100
fdc0             271147        0
--------------------------------
Total          72799752      100
```

- les principaux événements dans le système (*-s*). Voici un exemple de sortie :

```
> vmstat -s

        0 swap ins
        0 swap outs
        0 pages swapped in
        0 pages swapped out
184255741 total address trans. faults taken
  2309658 page ins
```

```
   545726 page outs
  5286844 pages paged in
  1587952 pages paged out
  2862764 total reclaims
  2827583 reclaims from free list
        0 micro (hat) faults
184255741 minor (as) faults
  1874375 major faults
 46464886 copy-on-write faults
 42803524 zero fill page faults
  4955194 pages examined by the clock daemon
       51 revolutions of the clock hand
  3336525 pages freed by the clock daemon
  1437031 forks
    80775 vforks
  1636403 execs
647112218 cpu context switches
555516593 device interrupts
358378561 traps
1108990825 system calls
335525799 total name lookups (cache hits 92%)
  2862097 toolong
 42843783 user   cpu
 32750241 system cpu
 62480945 idle   cpu
  6983879 wait   cpu
```

Outil général pour mesurer le système : SAR

Le principal utilitaire sur Solaris 2 pour mesurer le système est *sar*. L'utilitaire *sar* possède une quantité énorme d'options qui lui permettent d'obtenir toutes les statistiques du système. Il permet d'obtenir de multiples informations et de les emmagasiner dans des fichiers. *sar* peut même produire des moyennes sur des données amassées. *sar* peut donc fournir des informations sur :

- les tables du systèmes (*-v*) ;
- les accès aux fichiers (*-a*) ;
- l'utilisation des tampons du système (*-b*) ;
- les appels système (*-c*) ;
- les activités des unités d'entrées/sorties (*-y* pout terminal, *-d* pour les disques ou autres périphériques du même type) ;
- la mémoire (*-k* pour la mémoire du noyau, -r pour la mémoire inutilisée), la pagination (*-g* et *-p*) et le swapping (*-w*) ;

- la communication inter-processus (*-m*) ;
- les files d'attente du sysème (*-q*) ;
- l'utilisation de l'UCT (*-u*) ;

Outil général pour mesurer le système : **top**

Pour obtenir une lecture continue des activités du système, vous pouvez utiliser *top*. Il donne des informations sur tous les processus en exécution. En particulier, il fournit les informations suivantes sur les processus : la commande qui a généré le processus, l'usager qui a démarré le processus, sa priorité, sa taille, son état, son occupation courante de la mémoire et le temps UCT consommé.

Outil général pour mesurer le système : SymbEl et SE

Sun a développé un langage (SymbEL) ainsi qu'un interpréteur (SE) qui permettent de programmer plus facilement des outils pour contrôler la performance de Solaris. Plusieurs exemples sont aussi fournis avec la distribution. Ces outils utilisent des couleurs pour caractériser l'utilisation des ressources. Ces couleurs sont blanc (libre), bleu (non-balancé), vert (aucun problème), ambre (condition d'avertissement), rouge (surcharge) et noir (critique). Ces outils utilisent des règles précises pour déterminer ces couleurs [9]. Voici quelques exemples d'outils basés sur *SE* :

- *zoom.se*

  Cet utilitaire donne des informations sur le comportement du système. Il identifie les problèmes potentiels du système et donne des solutions pour les régler. Il donne une vue global de la performance du système ainsi qu'une vue de chaque ressource individuellement.

  Cet outil présente les résultats en utilisant un interface graphique et la couleur. Il utilise aussi plusieurs autres outils codés en *se*.

- *virtual_adrian.se*

  Adrian Cockcroft est le spécialiste de la performance chez SUN. Cet outil fait des mesures et applique des recommandations telles qu'elles seraient dictées par ce spécialiste. Les résultats sont similaires à ceux de l'outil *zoom.se*.

- *cpu_meter.se*

  Cet outil offre un interface graphique illustrant le pourcentage d'utilisation des UCTs.

- *collisions.se*

  Cet outil indique le pourcentage de collisions sur le réseau Ethernet.

- *cpg.se*

  Cet outil est un guide pour la planification de capacité. Il détecte les problèmes et conseille des modifications.

9. Nous avons déjà donné quelques exemples de règles. Les détails des règles se trouvent dans *Adian Cockcroft, System Performance Monitoring, http ://www.sun.com/950901/columns/adrian/column1.html.*

- *pure_test.se*

  Cet outil affiche les règles et l'état du système.

- *xit.se*

  Cet outil affiche les mêmes informations que *iostat* mais dans une fenêtre graphique.

- et plusieurs autres (voir /opt/RICHPse/examples/).

1.10 Exemples de charges de travail de tests

1. 007 (ODBMS)

   Description : Designed to simulate a CAD/CAM environment.

   Tests :

   - pointer traversals over cached data, disk-resident data, sparse traversals, and dense traversals

   - updates : indexed and unindexed object fields, repeated updates, sparse updates, updates od cached data and creation and deletion of objects

   - queries : exact-match lookup, ranges, collection scan, path-join, ad-hoc join, and single-level make

   Originator : University of Wisconsin

   Versions : unknown

   Availability of Source : free from ftp.cs.wisc.edu :/007

   Availability of Results : free from ftp.cs.wisc.edu :/007

   Entry Last Updated : Thu Apr 15 15 :08 :07 1993

2. AGE Test Suite

   A measure of X-Window system performance from AGE Logic Inc.

3. AIM Technology Benchmark

   A commercially controlled family of benchmarks.

   Suite II Thirty-six single-threaded measures of the timing of specific system functions.

   Suite III A mix of synthetic and real workloads that simulate a multiuser Unix environment. The benchmark measures user throughput and response time degradation under load.

   Performance Report Measures in the AIM performance report are derived using the Suite III benchmark :

   AIM overall performance, normalized with a VAX 11/780 equal to 1.0

   Users number of active users where response time becomes unacceptable

   throughput peak throughput with the optimum number of users

   utilities a measure of throughput for a mix of standard Unix utilities

   contact : Amy Yowell (800) 848-8649

4. bc

5. Bench++

   Bench++ is a standard set of C++ benchmarks. More information is available from http ://- paul.rutgers.edu/ orost/bench_plus_plus.html.

   Source is available from :

   http ://paul.rutgers.edu/ orost/bench_plus_plus.tar.Z

   ftp ://paul.rutgers.edu/pub/bench++.tar.Z

6. bonnie

   A I/O throughput benchmark developed by Tim Bray at Waterloo University. Bonnie measures filesystem performance under conditions designed to resemble operations on large text databases using a 100MB file.

   This is a file system benchmark that attempts to study bottlenecks - it is named 'Bonnie' for semi-obvious reasons.

   Specifically, these are the types of filesystem activity that have been observed to be bottlenecks in I/O-intensive applications, in particular the text database work done in connection with the New Oxford English Dictionary Project at the University of Waterloo.

   It performs a series of tests on a file of known size. By default, that size is 100 Mb (but that's not enough - see below). For each test, Bonnie reports the bytes processed per elapsed second, per CPU second, and the percent CPU usage (user and system).

   In each case, an attempt is made to keep optimizers from noticing it's all bogus. The idea is to make sure that these are real transfers to/from user space to the physical disk.

   contact : tbray@watsol.waterloo.edu

7. BYTE Unix Benchmark

   This is a benchmark suite similar in spirit to SPEC, except that it's smaller and contains mostly things like "sieve" and "dhrystone". If you are comparing different UN*X machines for performance, this gives fairly good numbers. Note that the numbers aren't useful for anything except (perhaps, as in "maybe") for comparison against the same bench- mark suite run on some other system.

   The BYTE Magazine Unix benchmarks were last updated in July, 1991 to version 3. The version 3 benchmark includes measurements of double precision arithmetic (dhrystone 2 with and without register variables), 7 arithmetic measures, system call overhead, process creation (fork and execl), file copy throughput, pipe throughput and context switching, and a recursive Tower of Hanoi.

8. Chalmers Workstation User's Benchmark Suite

   A suite of benchmarks developed in 1991 by Chalmers University of Technology in Gothenburg, Sweden.

9. CPU2

   The CPU2 benchmark was invented by Digital Review (now Digital News and Review). To quote DEC, describing DN&R's benchmark, CPU2 ...is a floating-point intensive series

of FORTRAN programs and consists of thirty-four separate tests. The benchmark is most relevant in predicting the performance of engineering and scientific applications. Performance is expressed as a multiple of MicroVAX II Units of Performance.

The CPU2 benchmark is available via ftp from swedishchef.lerc.nasa.gov in the drlabs/cpu directory. Get cpu2.unix.tar.Z for unix systems or cpu2.vms.tar.Z for VMS systems.

10. DBMS Labs Benchmark

    A benchmark produced in 1991 by DBMS Magazine to measure database server performance in a LAN-based client-server environment. Results are expressed as transactions per second for each of seven characteristic application mixes : accounting, analyst, batch reporting, data entry, financial, heavy insert, and sales support.

11. Debit-Credit benchmark

    Until recently Debit-Credit was the most common transaction processing benchmark. Debit-Credit is a stylized abstraction of the teller support system in a multi-branch bank. It was first described it the article "A Measure of Transaction Processing Power", anonymous, in the April 1985 issue of Datamation.

12. dhrystone

    A synthetic workload developed by R.P. Wecker in 1984. Dhrystone is patterned after the Whetstone benchmark but reflects a systems rather than scientific workload.

    available from netlib@ornl.gov ; "send index from benchmark"

13. Digital Review

14. EuroBen

    The main contact for EuroBen is Aad van der Steen.

    Name : Aad van der Steen email : actstea@cc.ruu.nl address : Academish Computercentrum Utrecht

    Budapestlaan 6

    3584 CD Utrecht

    The Netherlands

    phone : +31-30531444

    fax : +31-30-531633

15. Fhourstones

    Small integer-only program that solves positions in the game of connect-4 using exhaustive search with a very large transposition table. Written in C.

    Originator : John.Tromp@cwi.nl Versions : 1.0

    Availability of Source : ftp.nosc.mil :pub/aburto/c4.shar

    Availability of Results : ftp.nosc.mil :pub/aburto/c4.tbl

    –John Tromp (tromp@cwi.nl)

16. fsanalyze

17. FLOPS

    Estimates MFLOPS rating for specific FADD, FSUB, FMUL, and FDIV instruction mixes. Four distinct MFLOPS ratings are provided based on the FDIV weightings from 25operations. Works with both scalar and vector machines.

    The following data files are available from ftp.nosc.mil/pub/aburto :

    Source : flops20.c

    Result : flops_1.tbl, flops_2.tbl, flops_3.tbl, and flops_4.tbl

    –Alfred Aburto

18. FLOPS

19. gbench

    A measure of X-Window system graphics performance.

    available from uunet : comp.sources.unix ; volume15

20. Gabriel

    A LISP benchmark.

21. Gibson mix

    Not strictly a benchmark : J.C. Gibson of IBM in 1960 used dynamic instruction traces of program running on the IBM 650 and 704 computers to establish the relative frequency of each machine instruction. He then used an appropriately weighted average of individual instruction timings to compute ... MIPS ! (actually KIPS on those machines).

22. GPCmark

    Graphics Performance Count, a measure of graphics performance defined by the National Computer Graphics Association (NCGA) measures graphic system performance in terms of measures like polygons/second and vectors/second.

23. Hanoi

    An integer program that solves the Towers of Hanoi puzzle using recursive function calls.

    The following data files are available from ftp.nosc.mil/pub/aburto : Source : hanoi.c Result : hanoi.tbl

    –Alfred Aburto

24. Hartstone

    Hartstone is a benchmark for measuring various aspects of hard real time systems from the Software Engineering Institute at Carnegie Mellon.

    You can get this by anonymous ftp to ftp.sei.cmu.edu [128.237.2.179], in the pub/hartstone directory.

25. Heapsort

    An integer program that uses the "heap sort" method of sorting a random array of long integers up to 2 megabytes in size.

    The following data files are available from ftp.nosc.mil/pub/aburto : Source : heapsort.c Result : heapsort.tbl

    –Alfred Aburto

26. iobench

    IOBENCH is a multi-stream benchmark that uses a controlling process (iobench) to start, coordinate, and measure a number of "user" processes (iouser) ; the Makefile parameters used for the SPEC version of IOBENCHP cause ioserver to be built as a "do nothing" process.

27. iocall

28. iostone

    A I/O performance benchmark developed by Arvin Park at Princeton in 1986. It measures filesystem performance for a specific mix of I/O sizes and operations.

    contact : park@iris.ucdavis.edu

    available from : nbslib@cmr.ncsl.nist.gov ; "send index"

29. iozone

    A highly portable I/O performance benchmark by Bill Norcott that measures filesystem performance reading and writing sequential files with a variety of block sizes. This test writes a X MEGABYTE sequential file in Y byte chunks, then rewinds it and reads it back. [The size of the file should be big enough to factor out the effect of any disk cache.]. Finally, IOZONE deletes the temporary file

    The file is written (filling any cache buffers), and then read. If the cache is >= X MB, then most if not all the reads will be satisfied from the cache. However, if it is less than or equal to .5X MB, then NONE of the reads will be satisfied from the cache. This is becase after the file is written, a .5X MB cache will contain the upper .5 MB of the test file, but we will start reading from the beginning of the file (data which is no longer in the cache)

    In order for this to be a fair test, the length of the test file must be AT LEAST 2X the amount of disk cache memory for your system. If not, you are really testing the speed at which your CPU can read blocks out of the cache (not a fair test)

    IOZONE does not normally test the raw I/O speed of your disk or system. It tests the speed of sequential I/O to actual files. Therefore, this measurement factors in the efficiency of you machines file system, operating system, C compiler, and C runtime library. It produces a measurement which is the number of bytes per second that your system can read or write to a file.

    contact : norcott_bill@tandem.com

30. ipbench

    An image processing benchmark by Mark T. Noga at Lockheed. This benchmark measures the performance of 50 common image processing opeations on 512x512 images with 8-bit pixels. This benchmark emphasizes integer and boolean operations and program flow.

available from : nbslib@cmr.ncsl.nist.gov ; "send index"

31. Kenbus1

    Kenbus1 is half of the Systems Performance Evaluation Cooperative's System Development - Multitasking (SDM) benchmark. Kenbus1 is derived from the Monash University (Melbourne, Australia) suite of Unix benchmarks (musbus version 5.2) originally developed by Ken McDonell. The system under evaluation is used to execute increasing numbers of copies of a standard workload ("script"). Throughput is measured in scripts completed per hour, and both the throughput curve and peak value are reported. The script use some 18 Unix commands including cc, cat, grep, mkdir, and rm. The mix is designed to represent a Unix/C research and development environment.

32. Khornerstone

    A commercially controlled benchmark. Results are copyrighted and closely held by Workstation Laboratories.

    contact : (214) 570-7100

33. lhynestone

    A measure of performance for graphics systems.

34. Linpack

    A floating point benchmark (kernel benchmark) developed by Jack Dongarra at Los Alamos National Laboratories in 1979 which consist of linear algebra routines. Originally written and commonly used in Fortran ; a C version also exists. The benchmark consists of a series of Fortran kernels representing common liner algebra matrix operations on 100x100, 300x300 and 1000x1000 matrices. the results are in Millions of Floating Point Operations per Second (MFLOPS). Almost all of the benchmark's time is spent in a subroutine ("saxpy" in the single-precision version, "daxpy" in the double-precision version) doing the inner loop for frequent matrix operations : y(i) = y(i) + a * x(i)

    contact : dongarra@cs.utk.edu

    available from netlib@ornl.gov ; "send index from benchmark"

35. Livermore Loops (LFK)

    A floating point benchmark developed at Lawrence Livermore Laboratories. The benchmark consists of some 50 Fortran inner loops taken from various applications in use at the Labs in the early '80s.

36. Los Alamos benchmarks

    available from : nbslib@cmr.ncsl.nist.gov ; "send index"

37. Matrix Multiply (MM)

    This program (mm.c) contains 9 different algorithms for doing matrix multiplication (500 X 500 standard size). Results illustrate the enormous effects of cache thrashing versus algorithm, machine, compiler, and compiler options.

    The following data files are available from ftp.nosc.mil/pub/aburto :

Source : mm.c

Result : mm_1.tbl, and mm_2.tbl

38. mendez

39. McCalpin Kernels

    John McCalpin at the University of Delaware is collecting results of a series of Fortran kernels. These measure how well a machine's memory system can move large uncached contiguous structures around and the extent to which floating point operations can be overlapped with fetches from memory. The kernels measure time for copy, scale, add, and SAXPY operations (c=a, c=constant*a, c=a+b, c=a+constant*b).

40. mhawstone

    A synthetic benchmark developed by Jeff Mawhirter at Mead Data Central. It is intended to be more representative of a typical application than whetstone or dhrystone.

    contact : jeffm@meaddata.com

41. MIT Volume Stress Test

    A measure of X-Window system performance.

42. musbus

    The musbus benchmark was developed at and is available from Monash University in Melbourne, Australia. It simulates typical and heavy loads on general purpose Unix machines (i.e. servers rather than workstations). The benchmark produces accurate indications of process throughput and I/O bandwidth and can be used for kernel and configuration tuning. The current version is 5.2 and has been repackaged as the KENBUS1 benchmark in the SPEC SDM suite.

    contact : musbus@bruce.cs.monash.edu.au

    kenj@yarra.oz.au (to join newsletter mailing list)

    available from uunet : comp.sources.unix volume11/musbus (5.0)

    volume12/musbus5.2

    (upgrade kit from 5.0)

43. Netperf

    Netperf - a networking performance benchmark/tool. The current version includes throughput (bandwidth) and request/response (latency) tests for TCP and UDP using the BSD sockets API, DLPI, Unix Domain Sockets, the Fore ATM API, and HP HiPPI Link Level Access. Future versions may support additional tests for XTI/TLI-TCP/UDP, and WINSOCK ; in no particular order, depending on the whim of the author and public opinion. Included with the source code is a .ps manual, two manpages, and a number of example scripts.

    More information about netperf, and a database of netperf results can be found with a forms-capable WWW broser at the

    <A HREF="http ://www.cup.hp.com">Netperf Page</A>.

Various versions of netperf are also available via anonymous FTP from many locations, including, but not limited to :

ftp ://ftp.cup.hp.com/dist/networking/benchmarks

ftp ://col.hp.com/dist/networking/benchmarks

ftp ://ftp.sgi.com

ftp ://hpux.csc.liv.ac.uk (and mirrors)

44. NAS Kernels

    The sequential C versions of the NAS CFD (computational fluid dynamics) benchmarks - appbt and appsp - are now available from the anonymous ftp site ftp.cs.wisc.edu :/wwt/-Misc/NAS.

    This distribution contains the C version of the fortran cfd codes written at NASA Ames Research Center by Sisira Weeratunga. These codes were converted to C by a team of students for their class project for CS 838-3/ChE 562 offered in Spring 1993 by Mark D. Hill, Sangtae Kim and Mary Vernon at the University of Wisconsin at Madison. CS 838-3/ChE 562 was an experimental course that brought computer scientists and computation scientists together to promote interdisciplinary research.

    You should have a NAS license for the original fortran code. NASA has given us permission to distribute our "significant" changes freely.

    You can obtain the sequential fortran codes and a license by writing to :

    ATTN : NAS Parallel Benchmark Codes

    NAS Systems Division

    Mail Stop 2588

    NASA Ames Research Center

    Moffett Field

    CA 94035

45. NCR benchmark

46. Neil Nelson Business Benchmark

    The benchmark is a suite of eighteen synthetic workloads. These are used to measure computer performance by measuring the elapsed time to execute various numbers of copies of the workstream. Seven of the tests are summed to produce a CPU score and 9 are summed to produce a Disk score. This is a commercially controlled benchmark. Raw results are copyrighted and closely held.

47. nettest

    A network performance analysis tool developed at Cray Research. It is a derivative from ttcp. This benchmark measures throughput across several TCP and UDP connections at once.

48. NFSstone

    A measure of Network File System (NFS) performance standardized by 6 major NFS vendors in October, 1991. The NFSstone specification has been submitted to the Systems Performance

Evaluation Cooperative (SPEC) for inclusion in a SPEC benchmark suite. NFSstone is based on work by Auspex.

contact : Mike Bennett (408) 492-0090

49. nhfsstone

    A measure of Network File System (NFS) performance developed and maintained by Legato Systems. It measures server response time and server load (calls per second).

    The work in this area continued within the LADDIS group and finally within SPEC. The SPEC benchmark 097.LADDIS (SFS benchmark suite, see separate FAQ file on SPEC) is intended to replace Nhfsstone, it is superior to Nhfsstone in several aspects (multi-client capability, less client sensitivity).

    available from : ngfsstone-request@legato.com

    "send unsupported nhfsstone"

50. Nullstone

    The NULLSTONE Automated Compiler Performance Analysis Tool uses a QA approach of test coverage and isolation to measure an optimizer. The performance test suite is comprised of 6,500+ tests covering a wide range of compiler optimizations. The tool includes a report generator that generates performance reports, failure reports, regression reports, and competitive analysis reports. NULLSTONE runs on UNIX, Win3.1, Win95, WinNT, DOS, and MacOS.

    Additional information :

    Nullstone Corporaiton

    48531 Warm Springs Boulevard, Suite 404

    Fremont, CA 94555-7793

    Phone : (800) 995-2841 (international (510) 490-6222)

    FAX : (510) 490-9333

    email : info@nullstone.com

    www : http ://www.nullstone.com

    –Christopher Glaeser

51. Object Operations Benchmark

    A measure of database performance, emphasizing data retrieval and manipulation from a CAD perspective (i.e., lots of simple operations, few elaborate queries).

52. parcbench

    A benchmark written in C to measure performance of Unix System V shared-memory multiprocessor machines.

53. PC Bench/WinBench/NetBench

    See http ://www.ziff.com/ zdbop

PC Bench 9.0, WinBench 95 Version 1.0, Winstone 95 Version 1.0, MacBench 2.0, NetBench 3.01, and ServerBench 2.0 are the current names and versions of the benchmarks available from the Ziff-Davis Benchmark Operation (ZDBOp).

54. PERFECT

55. Performance Testing Alliance

    The Performance Testing Alliance is an industry group established in mid-1990 to standardize testing of Local Area Network performance.

56. Picture Level Benchmarks

    A set of benchmarks developed by the Graphics Performance Characterization group to characterize the performance of graphics display systems.

    contact : Bob Willis (NCGA) (703) 698-9600

57. plum benchmarks

    available from uunet : comp.sources.unix ; volume20

58. RAMP-C

    A synthetic OLTP benchmark used by IBM, originally developed in COBOL under CICS. RAMP-C measures transactions per second (TPS) ; each transaction averages 200,000 instructions and 19 I/Os.

59. Rendermark

60. Rhealstone

    A benchmark for performance elements critical to real-time multitasking systems. Rhealstone measures :

    - task switch time

    - preemption time

    - interrupt latency

    - semaphore related delays

    - deadlock break time

    - intertask message latency

    Rhealstone was first described in "Rhealstone : A Real-Time Benchmarking Proposal", Dr. Dobb's Journal, February, 1989.

61. RhosettaStone

    A benchmark for speech synthesis and speech recognization.

62. SEI ADA benchmark (Tm DOD)

    A set of ADA language benchmarks developed by the Software Engineering Institute at Carnegie-Mellon University.

63. Sieve of Eratosthenes

    An integer program that generates prime humbers using a method known as the Sieve of Eratosthenes.

    otis.stanford.edu :/pub/benchmarks/c/small/sieve.c

    ftp.nosc.mil :pub/aburto/nsieve.c

    ftp.nosc.mil :pub/aburto/nsieve.tbl

    sunic.sunet.se :/SRC/sec8/bench-dry/sieve.c

64. Sim

    An integer program that compares DNA segments for similarity.

    The following files are available from ftp.nosc.mil/pub/aburto : Source : sim.shar Result : sim.tbl

65. SLALOM

    Developed by Gustafson et.al. at Ames Lab, Iowa State University and described in "SLALOM : The First Scalable Supercomputer Benchmark", Supercomputing Review, November, 1990.

    available from archive at : tantalus.al.iastate.edu in /pub/Slalom

66. smith

67. SSBA

    The SSBA is the result of the studies of the AFUU (French Association of Unix Users) Benchmark Working Group. This group, consisting of some 30 active members of varied origins (universities, public and private research, manufacturers, end users), has assigned itself the goal of thinking on the problem of assessing the performance of data processing systems, collecting a maximum number of tests available throughout the world, dissecting the codes and results, discussing the utility, fixing versions and supplying them in the form of a magnetic tape with various comments and procedures.

    This tape is therefore both a simple and coherent tool for the end users and also for the specialists, providing a clear and pertinent initial approximation of the performance, and could also become a "standard" in the Unix (R) world. In this way the SSBA (Synthetic Suite of Benchmarks from the AFUU) originated and here you find release 1.21E.

    athene.uni-paderborn.de :/doc/magazin/ix/tools/ssba1.22.tar

    ftp.germany.eu.net :/pub/sysadmin/benchmark/ssba/ssba.shar.Z

    ftp.inria.fr :/system/benchmark/SSBA/ssba1.22E.tar.Z

    ftp.inria.fr :/system/benchmark/SSBA/ssba1.22F.tar.Z

    ftp.inria.fr :/system/benchmark/SSBA/ssba-syntheses.tar.Z

68. System Development Throughput

    Software Development Throughput (SDeT) is half of the Systems Performance Evaluation Cooperative's System Development - Multitasking (SDM) benchmark. SDeT was developed

from a proprietary AT&T benchmark by the University of California at Berkeley. The system under evaluation is used to execute increasing numbers of copies of a standard workload ("script"). Throughput is measured in scripts completed per hour, and both the throughput curve and peak value are reported. The script use some 150 Unix commands including spell, nroff, diff, make, and find. The mix is designed to represent a C-based software development environment.

69. SPECmark

    SPEC stands for "Standard Performance Evaluation Corporation", a non-profit organization with the goal to "establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers" (from SPEC's bylaws). The SPEC benchmarks and more information can be obtained from

    SPEC [Standard Performance Evaluation Corporation]

    10754 Ambassador Drive, Suite 201

    Manassas, VA 22110, USA

    USA

    Phone : +1-703-331-0180

    Fax : +1-703-331-0181

    E-Mail : spec-ncga@cup.portal.com

    The current SPEC benchmark suites are

    CINT95 CPU intensive integer benchmarks) together :

    CFP95 CPU intensive floating point benchmarks) SPEC95

    SDM UNIX Software Development Workloads

    SFS System level file server (NFS) workload

    The old CPU benchmark suites

    CINT92 (CPU intensive integer benchmarks)) together :

    CFP92 (CPU intensive floating point benchmarks)) SPEC92

    will cease to be supported by SPEC in 1996.

    See separate FAQ file on SPEC benchmarks.

    –Reinhold Weicker

    The following data files are available from ftp.nosc.mil/pub/aburto :

    specin89.tbl specfp89.tbl speccorr.tbl

    specin92.tbl specft92.tbl –Alfred Aburto

70. SPEC SDM 1.0

    System Development - Multitasking is the Systems Performance Evaluation Cooperative's second benchmark. It consists of two measures : Software Development Throughput (SDeT), measuring performance in program development applications, and Kenbus1, measuring performance in research and development applications.

71. Stanford

    - 1988

    - Stanford University (J. Hennessy, P. Nye)

    - C

    - comparison of RISC and CISC

    - contains the 2 modules Stanford Integer and Stanford Floating Point

    Stanford Integer :

    - 8 little applications (integer matrix mult., sorting alg. (quick, bubble, tree), permutation, hanoi, 8 queens, puzzle)

    Stanford Floating Point :

    - 2 little applications (FFT, matrix mult.)

    The characteristics of the programs vary, but most of them have array accesses. There seems to be no official publication (only a printing in a performance report). Secondly, there is no defined weighting of the results (Sun and MIPS compute the geometric mean).

    Survey of Benchmarks, E. R. Brocklehurst.

    A Benchmark Tutorial, W. Price.

    "A detailed look at some popular benchmarks", R. P. Weicker.ford Small Programs Benchmark Set

72. Stream

    STREAM is a synthetic benchmark which measures sustainable memory bandwidth with and without simple arithmetic, based on the timing of long vector operations. STREAM is available in Fortran and C versions, and the results are used by all major vendors in high performance computing.

    A discussion of the benchmark and results on some 200 system configurations are available at :

    http ://perelandra.cms.udel.edu/hpc/stream/

    ftp ://perelandra.cms.udel.edu/bench/stream/

    Contact : John D. McCalpin, mccalpin@udel.edu,

    http ://perelandra.cms.udel.edu/ mccalpin

73. SYSmark

    July 28, 1993, Santa Clara, Calif.–The Business Applications Performance Corp. (BAPCo) announces SYSmark93 for Windows and SYSmark93 for DOS, benchmark software that provides objective performance measurement based on the world's most popular PC applications and operating systems. A third program, SYSmark93 for Servers, is due for release by BAPCo in the third quarter of 1993.

    SYSmark93 provides benchmarks that can be used to objectively measure performance of IBM PC-compatible hardware for the tasks users perform on a regular basis. The benchmarks are comparative tools for those who make purchasing decisions for anywhere from 10 to a

thousand or more PCs. SYSmark93 has been endorsed by the BAPCo membership, which includes the world's leading PC hardware and software vendors, chip manufacturers, and industry publications.

SYSmark93 benchmarks represent the workloads of popular programs in such applications as word processing, spreadsheets, database, desktop graphics and software development. Benchmarking can be conducted on the user's own system or at a vendor's site using the standards set by BAPCo to ensure consistency of the results.

SYSmark93 for Windows is for those interested in evaluating systems-level performance of PCs running Microsoft Windows applications. The program features a new Windows based workload manager, scripts for 10 Windows applications, automation tools, and a disclosure report generator. SYSmark93 for DOS, an upgrade to SYSmark92, is aimed at those interested in evaluating systems-level performance of PCs running DOS applications only. Both programs can generate performance metrics in three ways : as a composite of all the different applications ; for a specific category of applications, such as word processing or spreadsheets ; or for individual software programs.

Workloads based on the following applications are included in SYSmark93 for Windows and SYSmark93 for DOS :

SYSmark93 for Windows

WORD PROCESSING

Word for Windows 2.0b

WordPerfect for Windows 5.2

AmiPro 3.0

SPREADSHEETS

Excel 4.0

Lotus 1-2-3 for Windows 4.0

DATABASE

Paradox for Windows 1.0

DESKTOP GRAPHICS

CorelDraw 3.0

DESKTOP PRESENTATION

Freelance Graphics for Windows 2.0

PowerPoint 3.0

DESKTOP PUBLISHING

PageMaker 5.0

SYSmark93 for DOS

WORD PROCESSING

WordPerfect 5.1

SPREADSHEETS

Lotus 1-2-3 3.4

QuattroPro 4.0

DATABASE

Paradox 4.0

dBASE IV 1.5

DESKTOP GRAPHICS

Harvard Graphics 3.0

SOFTWARE DEVELOPMENT

Borland C++ 3.1

Microsoft C 6.00

SYSmark93 for Windows and SYSmark93 for DOS will be available in August from BAPCo for $390 each. Licensed SYSmark92 users will be able to upgrade to either SYSmark93 for Windows or SYSmark93 for DOS for $99 each.

A non-profit corporation, BAPCo's charter is to develop and distribute a set of objective performance benchmarks based on popular computer applications and industry standard operating systems. Current BAPCo members include Adaptec, Advanced Micro Devices, AER Energy Resources, Apricot Computers, Chips and Technologies, Compaq, Cyrix, Dell, Digital Equipment Corp., Gateway2000, Epson, Hewlett-Packard, IBM, Infoworld, Intel, Lotus, Microsoft, NCR, Unisys and Ziff-Davis Labs.

FOR MORE INFORMATION :

John Peterson, BAPCo

Phone : 408-988-7654, email : John_E_Peterson@ccm.hf.intel.com

Bob Cramblitt, Cramblitt & Company

Phone : 919-481-4599, Fax : 919-481-4639

74. tbench

    available from uunet : comp.sources.misc

75. TFFTDP

    This program performs FFT's using the Duhamel-Hollman method for FFT's from 32 to 262,144 points in size.

    The following data files are available from ftp.nosc.mil/pub/aburto :

    Source : tfftdp.c

    Result : tfftdp.tbl

    –Alfred Aburto

76. TPC

    The TPC (Transaction Processing Council) was established in 1988 by a group of major database vendors to standardize transaction processing benchmarks. Until now it has created three benchmarks.

- TPC A
  TPC A is a formalization by the Transaction Processing Council of the Debit-Credit benchmark which was first published in DATAMATION in 1985. It is based on a single, simple, update-intensive transaction which performs three updates and one insert across four tables. Transactions originate from terminals, with a requirement of 100 bytes in and 200 bytes out. There is a fixed scaling between tps rate, terminals, and database size. TPC-A requires an external RTE (remote terminal emulator) to drive the SUT (system under test).
  TPC-A differs from Debit-Credit in the following (although Debit-Credit was frequently honored in the breach) :
  - exponential arrival times
  - ten second think time (implies 10 terminals per TPS)
  - 90
  - allows LAN or X.25 connections
  - response time is measured at the terminal
  - formal requirements for atomicity, consistency, isolation, and durability
  TPC-A results must include a rather elaborate disclosure report, including configuration and environment. Results must include both transactions per second and total 5-year cost per transaction per second.
- TPC B
  TPC B is a formalization by the Transaction Processing Council of the TP1 benchmark. TPC-B uses the same transaction profile and database schema as TPC-A, but eliminates the terminals and reduces the amount of disk capacity which must be priced with the system. TPC-B is significantly easier to run because an RTE is not required. SO, TPC B is a batch benchmark and does not include user think time or communications overhead. Transactions are produced as rapidly as possible by a "generator".
- TPC-C
  TPC-C is completely unrelated to either TPC-A or B. TPC-C tries to model a moderate to complex OLTP system. The benchmark is conceptually based on an order entry system. The database consists of nine tables which contain information on customers, warehouses, districts, orders, items, and stock.
  The system performs five kinds of transactions : entering a new order, delivering orders, posting customer payments, retrieving a customer's most recent order, and monitoring the inventory level of recently ordered items.
  Transactions are submitted from terminals providing a full screen user interface. (The spec defines the exact layout for each transaction.)
  TPC-C was specifically designed to address many of the shortcomings of TPC-A. I believe it does this in many areas. It exercises a much broader cross-section of database functionality than TPC-A. Also, the implementation rules are much stricter in critical areas such as database transparency and transaction isolation. Overall, TPC-C results will be a much better indicator of RDBMS and OLTP system performance than previous TPC benchmarks.

Conclusion : - 1988 - Transaction Processing-performance Council, San Jose - non-profit corporation of 44 sw- and hw-companies to define transaction processing and database benchmarks

- Cobol - Contents (Basic system OLTP, Business appl. services, Databases, Complex data and Real time appl.) - 4 Benchmark-components (Data-, Database-, System and Query-model) - 4 Suites : TPC-A, ..., TPC-D

Suite TPC-A : On-line transaction processing of a database env. - measures performance in update-intensive database env. (OLTP) - result in transactions/sec - 2 metrivcs (local and wide area networks)

Suite TPC-B : Database benchmark - Database throughput (transactions/sec) - no OLTP - Suite TPC-C : Order-Entry benchmark (OLTP env.) - Business application services (Order-entry, Inventory Control, customer support, accounting)

Suite TPC-D : Decision-Support benchmark (OLTP env.) - Database stress test, simulation of a large database, complex queries

See :

Shanley Public Relations, Complete TPC Results, Performance Evaluation Review, Vol. 19 #2, Aug. 91 (14-23)

Shanley Public Relations, Complete TPC Results, Performance Evaluation Review, Vol. 19 #3, Feb. 93 (32-35)

77. ttcp

    The most commonly used measure of TCP/IP performance. Measures throughput on a single TCP or UDP circuit. Results are not verified or audited, and like TP1 the benchmark is frequently "enhanced".

    Ttcp times the transmission and reception of data between two systems using the UDP or TCP protocols. It differs from common “blast” tests, which tend to measure the remote inetd as much as the network performance, and which usually do not allow measurements at the remote end of a UDP transmission.

    available from archive at : sgi.com in /sgi/src/ttcp.shar

78. University of Wisconsin benchmarks

    A measure of relational database performance, said to be a good indicator of join performance.

79. U.S. Steel

    A set of COBOL business kernels assembled by U.S. Steel in 1965. Results are relative to the performance of an IBM 1460 ; modern mainframes score well in excess of 5,000.

80. VGX benchmark

81. Whetstone

    This is a synthetic workload representing a mix of floating point, procedure call, transcendental functions, etc. that make up a typical scientific workload. It is based on statistics gathered at National Physical Lab in England, using an Algol 60 compiler which translated Algol into instructions for the imaginary Whetstone machine. The compilation system was named after the small town Whetstone outside the City of Leicester, England, where it was designed. The original Whetstone benchmark was written in Algol by Curnow and Wichman at the U.K.’s National Physical Laboratory in Whetstone, England. The benchmark is described in

"A Synthetic Benchmark", Computer Journal, February, 1976. Although both Fortran and C versions exist the whetstone code is relatively old and fits entirely in cache on most current machines.

available from netlib@ornl.gov ; "send index from benchmark"

82. Workstation Laboratories benchmark

    Workstation Laboratories measures multiuser performance using a transaction processing benchmark similar to TP1. The Workstation Laboratories benchmark is modeled after TP1 but is written entirely in C (for portability). This benchmark is disk- intensive.

83. WPI benchmark suite

    A set of benchmarks developed by Worcester Polytechnic Institute Mach Research Group to compare the performance of Unix implementations. The suite consists of five synthetic workloads simulating gcc, a client-server database application, file backup, FTP transfer, and X-windows client-server activity. It also includes one real benchmark which solves a mathematical model of a jigsaw puzzle.

    contact : mach@cs.wpi.edu

84. x11perf

    Actually a performance monitor for MIT's X-Windows system rather than a benchmark, but frequently used to gather performance information anyway.

85. xbench

86. Xlib Protocol Test Suite

87. xstone

    netcom.com :/pub/micromed/uploads/xstones.summary.z

88. ZQH benchmark

    This benchmark measures the run time for a computational fluid dynamics code written in Fortran simulating one physical time step for unsteady blood low near the valve leaflet in the Penn State artificial heart. The program is partially vectorizable and requires some 30,000 million floating-point operations (MFLOP) to complete. Results run from 2 minutes on a Cray-YMP to 20 hours on a Sun-3.

Bibliographie

[1] Crunchbase : Bgs systems. https://www.crunchbase.com/organization/bgs-systems-2, 1998.

[2] Clang documentation : clang - the clang c, c++, and objective-c compiler. https://clang.llvm.org/docs/CommandGuide/clang.html, 2022.

[3] CPP C++ Papyrus : Cpp/c++ compiler flags and options. https://caiorss.github.io/C-Cpp-Notes/compiler-flags-options.html, 2021.

[4] Trademarkia : Best/1 trademark information. https://www.crunchbase.com/organization/bgs-systems-2, 1996.