Le projet SATURNE

Y. Deswarte, J.C. Fabre, J.C. Laprie, David Powell

INRIA
CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. : (1) 39 63 55 11

Rapports de Recherche

N° 445

LE PROJET SATURNE

Yves DESWARTE
Jean-Charles FABRE
Jean-Claude LAPRIE
David POWELL

Octobre 1985

Laboratoire d'Automatique
et d'Analyse des Systèmes
L. A. A. S. /C. N. R. S.
7, avenue du Colonel Roche
31077 Toulouse Cédex

Institut National de Recherche
en Informatique et Automatique
I. N. R. I. A.
Domaine de Voluceau-Rocquencourt
B. P. 105 - 78153 Le Chesnay Cédex

Le Projet SATURNE

	S	ystème
rép	A	rti
	T	olérant
les fa	U	tes et
les int	R	usions
volo	N	taires
ou accid	E	ntelles

Yves DESWARTE (INRIA)
Jean-Charles FABRE (INRIA)
Jean-Claude LAPRIE (LAAS-CNRS)
David POWELL (LAAS-CNRS)

Septembre 1985

Rapport de Recherche
LAAS N°85.222

Rapport de Recherche
INRIA N°445

RÉSUMÉ

Ce rapport présente les objectifs du projet SATURNE, projet commun entre le LAAS/CNRS et l'INRIA, lancé en 1984. Ce projet vise à apporter des solutions nouvelles aux besoins des sytèmes informatiques répartis, en matière de fiabilité par la tolérance aux fautes, et en matière de confidentialité par la tolérance aux intrusions.

Après une courte analyse du marché de la sûreté de fonctionnement et de la confidentialité, le rapport décrit une méthode originale de tolérance aux fautes : la saturation. La saturation consiste à occuper chaque site du réseau en lui faisant exécuter un exemplaire d'un processus actif. La tolérance aux fautes est réalisée par masquage des erreurs, au moyen de votes majoritaires sur les messages émis par les différents exemplaires du processus.

Pour ce qui concerne la confidentialité, une méthode originale de tolérance aux intrusions est décrite : la fragmentation-dissémination. La méthode consiste à découper les fichiers en fragments, à copier ces fragments en plusieurs exemplaires, et à éparpiller ces copies sur des sites d'archivage spécialisés.

ABSTRACT

This report describes the aims of the SATURNE project. This is a joint LAAS/CNRS - INRIA project; it has been initiated in 1984. The SATURNE project is aimed at exploring new solutions to distributed computing systems reliability by means of fault-tolerance, and security by means of intrusion-tolerance.

First, the report presents a short analysis of the market of dependable systems and data security. Then, a new fault-tolerance method is described : the saturation, which consists in keeping every site of the network busy, through running a copy of an active process. The messages sent by all of the copies of the process are voted, in order to achieve error-masking.

In order to achieve data security, a new intrusion-tolerance method is presented : the fragmentation-dissemination. This method consists in cutting the files into fragments, in copying these fragments in several copies, and in disseminating these copies on specialized archive sites.

SOMMAIRE

INTRODUCTION

Le développement spectaculaire des réseaux locaux se porte aujourd'hui essentiellement dans trois domaines :

- les réseaux industriels de commande-contrôle, qui épousent la répartition géographique des systèmes à contrôler : processus, ateliers flexibles, ...
- les réseaux informatiques d'ingéniérie (CAO, PAO, ...), faisant communiquer des stations de travail et des ordinateurs puissants, améliorant la productivité et la qualité du travail des concepteurs, la souplesse et l'efficacité des moyens de calcul, de stockage, d'entrée-sortie, ...
- les réseaux de bureautique, qui facilitent la communication interne des entreprises tout en améliorant les conditions de travail et la productivité du personnel.

Chacun de ces domaines pose ses propres problèmes et impose ses propres exigences. Pourtant, tous partagent le même besoin : la sûreté de fonctionnement du système informatique, qui peut être définie comme "la qualité du service qu'il délivre, qualité telle que les utilisateurs du système puissent lui accorder une confiance justifiée" [Laprie 84]. Parmi les atteintes à la sûreté de fonctionnement d'un système informatique, nous distinguerons les fautes (accidentelles) et les intrusions (volontaires).

Il apparaît de plus en plus nécessaire que les systèmes informatiques répartis soient à la fois fiables (par rapport aux fautes) et sûrs (par référence aux intrusions). Pour explorer certaines solutions possibles à ce problème, le L.A.A.S. et l'I.N.R.I.A. ont créé un projet commun : SATURNE. Parmi les solutions proposées, nous nous intéresserons ici plus particulièrement à deux techniques originales : la saturation, qui vise à tolérer les fautes, et la fragmentation-dissémination dont l'objectif est de tolérer les intrusions.

Si, par la suite, les mécanismes de la saturation et de la fragmentation-dissémination seront présentés séparément, c'est par souci didactique. Il ne faudrait pas en conclure que ces mécanismes sont indépendants. Au contraire, ils visent à répondre à un besoin global de tolérance aux fautes et de confidentialité des systèmes répartis; ils doivent être mis en oeuvre sur une même architecture matérielle; enfin, les algorithmes de saturation et de dissémination sont tout à fait similaires, dans leurs fonctions et dans leurs principes et utiliseront les mêmes outils et méthodes de développement et de vérification. De plus, le service que nous voulons offrir aux utilisateurs doit être transparent, aussi bien pour ce qui concerne les exécutions de processus en exemplaires multiples, que pour ce qui concerne la fragmentation et la dissémination des fichiers.

Le présent rapport comporte cinq parties :

- la première présente les tendances du marché des systèmes informatiques sûrs de fonctionnement,
- la deuxième discute des concepts de tolérance aux fautes et aux intrusions,
- la troisième expose les objectifs du projet SATURNE,
- la quatrième développe la technique de la saturation,
- la cinquième est dédiée à la fragmentation-dissémination,

I. LES TENDANCES DU MARCHÉ DE LA SURETÉ DE FONCTIONNEMENT

Selon une étude de marché de Frost & Sullivan de New-York [Frost 85], le marché des sytèmes tolérant les fautes devrait croître de 790 millions de $ en 1984 à 7,4 milliards de $ en 1988, soit une croissance moyenne de 70% par an, ce qui est sans doute le record dans le domaine informatique pour cette période. Un taux de croissance encore plus grand devrait être atteint par les systèmes répartis tolérant les fautes. L'importance de ce marché a provoqué la création d'une quantité de sociétes, principalement américaines, visant ce créneau : en dehors de Tandem qui détient environ les deux tiers du marché, citons Stratus, avec qui IBM a passé récemment des accords, et August, fondée par John Wensley (le concepteur de SIFT).

Pourtant, l'industrie française n'a pas encore suivi ce mouvement, et s'est cantonnée à des réalisations spécifiques dans le domaine des télécommunications (E10), dans le contrôle des centrales nucléaires (Controbloc), dans le domaine spatial (ARMURE), ou dans le domaine militaire. Mais on chercherait en vain un système français tolérant les fautes à usage général qui soit commercialisé. Pourtant, ce domaine a bénéficié d'un soutien important de la recherche française : si l'on se réfère au nombre de communications acceptées aux Fault-tolerant Computing Symposiums depuis leur création en 1971, la France se situe au 2ème rang, loin derrière les U.S.A., mais assez loin devant le Japon, encore que cette différence tende à se réduire dans les dernières années.

Ceci nous conduit à affirmer que cet effort de la recherche française dans le domaine doit être poursuivi, voire amplifié. Les systèmes répartis et la confidentialité sont parmi les axes prioritaires à encourager.

Pour ce qui concerne les systèmes répartis, de nombreuses études ont porté ces dernières années sur les mécanismes (à faible redondance) de détection et de recouvrement des erreurs, en particulier les mécanismes de reprise, avec les travaux associés sur l'atomicité des traitements (transactions), sur l'ordonnancement des processus, mais aussi sur les protocoles de communication dont la complexité croît au fur et à mesure que l'on prend en compte des classes de fautes plus larges. Ces études ont montré les limites de ces mécanismes, qu'il s'agisse de leur efficacité, de leur complexité, ou encore de leurs implications sur les logiciels d'application. Ce choix d'une faible redondance se justifiait lorsque le coût du matériel était prépondérant devant celui du surcoût du logiciel, et lorsque les utilisateurs n'étaient pas convaincus de la nécessité de tolérer les fautes. Ces deux critères ont suffisamment évolué pour qu'on puisse aujourd'hui se préoccuper davantage de simplicité et de transparence des mécanismes vis-à-vis du logiciel, plutôt que de coût du matériel. On peut donc envisager plus sereinement des systèmes à forte redondance permettant de masquer les erreurs au lieu de les corriger, aussi bien pour les erreurs de transmission que pour les erreurs de traitement. Ces mécanismes de masquage étant totalement transparents pour les logiciels d'application, ils sont d'autant plus intéressants dans des contextes de forts parallélisme et concurrence, ce qui semble être l'une des voies privilégiées du développement de l'informatique.

Pour ce qui concerne la confidentialité, là encore le marché croît en fonction du développement des applications à grande diffusion (minitels, bases de données, services bancaires électroniques, ...). D'importantes études et développements sont en cours visant à l'évitement des intrusions, par l'authentification (cartes à mémoire, ...) ou la protection. Mais peu de travaux (trop peu diront certains) sont menés dans le domaine de la tolérance aux intrusions, même si de nouveaux mécanismes cryptographiques sont apparus ces dernières années aux Etats-Unis.

C'est donc en parfaite harmonie avec ces tendances du marché que nous avons choisi comme objectifs du projet SATURNE la tolérance aux fautes et la confidentialité dans les systèmes répartis.

II. TOLÉRANCE AUX FAUTES ET TOLÉRANCE AUX INTRUSIONS

II.1 Tolérance aux fautes

Pour réaliser des systèmes sûrs de fonctionnement, on utilise en général deux techniques complémentaires :

- l'évitement des fautes,
- la tolérance aux fautes.

L'évitement des fautes consiste à utiliser des composants et des processus de fabrication de grande qualité, avec des vérifications suffisantes pour rendre l'apparition de fautes aussi peu probable qu'on le souhaite pour la durée de vie du système. Cependant, aussi peu probables que soient ces fautes, au fur et à mesure que croissent l'importance et la complexité des applications réparties sur les réseaux locaux, les conséquences des fautes s'aggravent, qu'il s'agisse de fautes de traitement ou de fautes de transmission, provoquant une corruption des données ou une interruption de service. Dans les applications les plus critiques ces conséquences peuvent devenir insupportables : on exige alors que le traitement soit mené à bien et que les résultats soient transmis dans des délais connus, même en cas de défaillance d'un ou plusieurs éléments de traitement ou de transmission : le système doit être tolérant aux fautes.

Il existe deux techniques pour tolérer les fautes :

- la détection et recouvrement des erreurs,
- la compensation des erreurs.

La première technique consiste à exécuter les traitements et les communications avec juste assez de redondance pour détecter les erreurs et pour être capable, en cas d'erreur, de rétablir le fonctionnement correct du système. Pour la détection, on peut utiliser des mécanismes matériels ou logiciels de contrôle de vraisemblance (parité, chien-de-garde, protection, test de bon fonctionnement, ...). Pour le recouvrement, on utilise des moyens de diagnostic, de reconfiguration, de réinitialisation ou de reprise. Cette technique est d'une mise en oeuvre difficile dans un environnement réparti où le parallélisme et la concurrence/coopération sont importants. En effet, les mécanismes de détection n'ont généralement pas une couverture de 100%, ce qui signifie que les erreurs peuvent se propager

et contaminer des éléments sains. Les mécanismes de recouvrement sont alors complexes et relativement inefficaces.

Au contraire, la technique de compensation des erreurs nécessite que les traitements et les communications soient exécutés avec suffisamment de redondance pour permettre de délivrer un résultat correct même en présence de fautes : les données présentent une redondance suffisante pour permettre de corriger les erreurs sans nécessiter d'autre information. Les codes correcteurs permettent une compensation par correction (en cas d'erreur, un algorithme de correction est mis en oeuvre). Le vote majoritaire est une méthode de compensation par masquage: les données erronées (si elles sont minoritaires) sont éliminées au profit des données majoritaires. Le vote a lieu systématiquement, qu'il y ait ou non errreur. Cette technique de masquage est très bien adaptée aux systèmes répartis, puisqu'elle empêche la propagation des erreurs. Son principal inconvénient est le surcoût mis en jeu par la redondance même en abscence de faute. Par exemple, sur un réseau point-à-point, un vote majoritaire sur N copies d'un processus conduit à échanger de l'ordre de N^2 messages pour chaque message fonctionnel (figure 1).

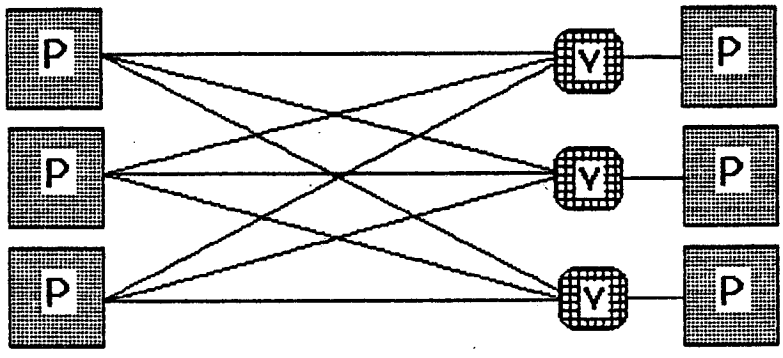


Figure 1
Exemple de vote majoritaire
(Réseau point-à-point)

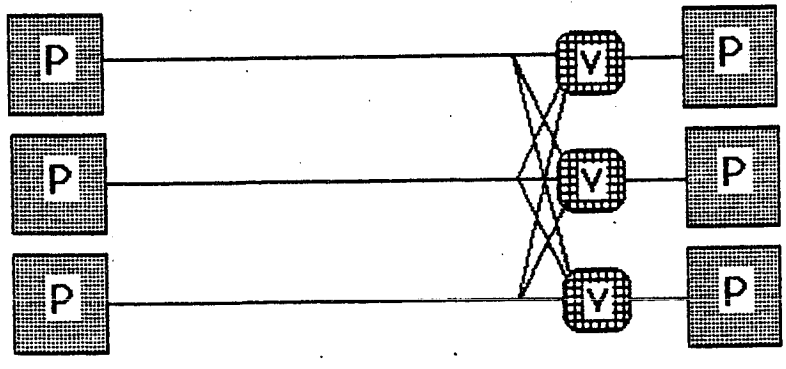


Figure 2
Exemple de vote majoritaire
(Réseau à diffusion)

L'efficacité des mécanismes de vote peut être améliorée par l'utilisation de communication à diffusion. Dans l'exemple de la figure 2, le nombre de messages émis est égal au nombre N de sites émetteurs.

Quand on conçoit un système tolérant les fautes, il faut aussi prendre en compte la notion de latence d'erreur. La latence d'erreur peut se définir comme le délai entre l'occurence de la faute et l'apparition de l'erreur résultante (pour une définition précise des fautes et erreurs, on se réfèrera à [Laprie 84]). De nombreuses études [Castillo 81, Iyer 82, ...] montrent la forte corrélation entre la probabilité d'apparition des erreurs et la charge du système, aussi bien pour les fautes logicielles que matérielles. L'explication la plus vraisemblable est que les fautes matérielles qui surviennent lors de périodes de faible activité ont une forte probabilité de se manifester sous forme d'erreurs effectives lorsque le système est fortement chargé. De la même facon les fautes de conception de logiciel ont une plus forte probabilité d'être sensibilisées lorsque la charge du système est plus importante. En fait, c'est lorqu'on a le plus besoin de puissance du système qu'il a le plus de chance d'être défaillant! Il n'est nul besoin d'évoquer la loi de Murphy pour expliquer ce phénomène. Réduire le temps de latence contribue donc à améliorer la confiance qu'on peut avoir dans le système.

Dans SATURNE, nous nous proposons d'expérimenter une technique originale de tolérance aux fautes que nous avons appelée saturation (d'où une justification du nom du projet : SATURNE = SATURation NEtwork = réseau à saturation). Le principe de la saturation consiste à créer de multiples copies de tous les processus actifs dans le système de façon à satisfaire les exigences suivantes :

- chaque site du réseau exécute un exemplaire d'un processus,
- les processus les plus critiques sont exécutés avec un plus grand nombre d'exemplaires,
- le masquage est obtenu par vote majoritaire sur les données échangées et les informations rémanentes.

Bien sûr, le nombre d'exemplaires d'un processus actif varie dynamiquement en fonction des créations ou des terminaisons des autres processus. Cette technique présente deux avantages importants :

- à tout instant le degré de redondance de chaque processus est maximal, et fonction du nombre de sites disponibles dans le système ; l'efficacité des mécanismes de masquage est maximale,
- puisque chaque site exécute un exemplaire d'un processus, les erreurs peuvent être détectées dès l'apparition de la faute : la latence d'erreur est minimale, aussi bien pour les fautes logicielles que matérielles.

Notons que la technique de saturation n'exclut pas l'usage de techniques de tolérance aux fautes logicielles telles que la programmation à versions multiples [Avizienis 85] : les exemplaires des processus pourraient exécuter des versions différentes de code, à condition d'offrir des interfaces identiques.

Notre approche par saturation présente des similitudes avec celle du WORM imaginée par une équipe de Xerox [Shoch 82], encore que cette étude ne visait pas la tolérance aux fautes, mais seulement l'évaluation de la répartition des processus et de leurs communications.

II.2 Tolérance aux intrusions

Alors que se développent de plus en plus d'applications "sensibles" (c'est-à-dire concernant des informations confidentielles), l'augmentation du nombre d'utilisateurs des réseaux locaux ou à grande échelle rend de plus en plus difficile et de moins en moins efficace l'usage de mécanismes d'évitement des intrusions, tels que l'authentification, le contrôle d'accès, la protection, ... Par analogie avec les techniques d'évitement et de tolérance aux fautes, on est amené à promouvoir, en complément de l'évitement, des techniques de tolérance aux intrusions : plutôt que d'empêcher un intrus malicieux ou involontaire d'accéder à des données confidentielles, il est préférable de rendre ces informations incompréhensibles pour tout individu non autorisé. La cryptographie est un exemple classique de tolérance aux intrusions. Parmi les inconvénients de la cryptographie, il faut citer un surcoût qui peut être important en traitement et en encombrement mémoire. De plus, son efficacité n'est pas totale : avec les techniques actuelles, un intrus disposant de suffisamment de puissance de calcul et de temps pourrait décoder un fichier crypté dont il aurait, par exemple, volé le support.

La technique de tolérance aux intrusions que nous voulons évaluer dans SATURNE est appelée "fragmentation-dissémination". Une technique analogue a d'abord été proposée pour les réseaux maillés [Koga 82] : chaque message transmis sur le réseau est découpé en fragments envoyés au noeud récepteur par différentes routes, avec une certaine redondance. Seul le noeud récepteur reçoit tous les fragments et lui seul est capable de reconstituer le message initial (figure 3).

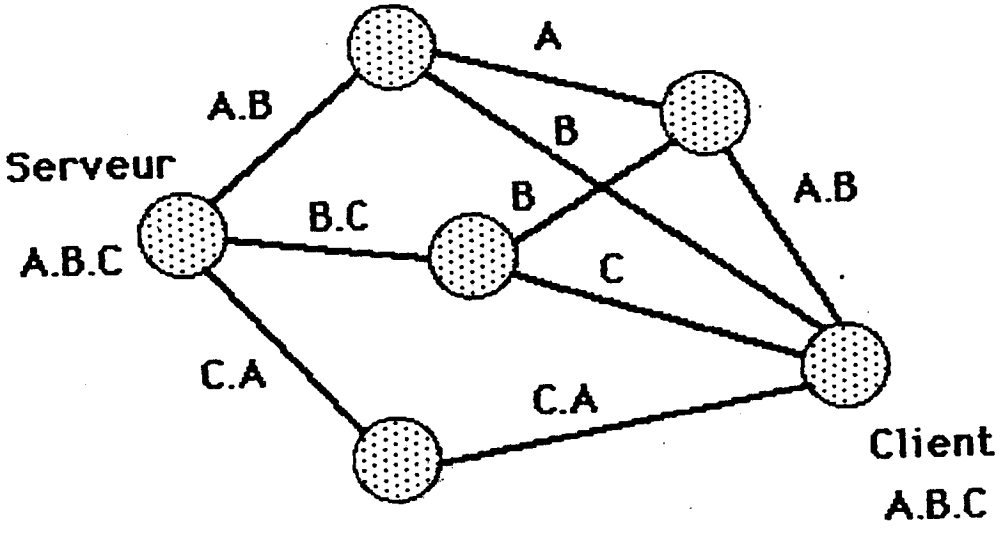


Figure 3
Fragmentation-dissémination sur réseau maillé

Une technique similaire peut être mise en oeuvre sur des réseaux locaux en bus ou en anneaux : chaque exemplaire d'un processus n'émettrait qu'un fragment du message, les différents fragments issus des différents exemplaires étant redondants entre eux; la fragmentation pourrait être réalisée de sorte que seuls les sites destinataires puissent reconstituer les messages, et voter sur les messages. Notre objectif est d'étendre ce

principe de fragmentation-dissémination au stokage des fichiers sur des sites spécialisés : pour cela, les fichiers logiques seraient fragmentés et ces fragments seraient éparpillés sur différents sites d'archivage [Fraga 85] (figure 4). Ainsi, les informations contenues dans un site d'archivage ne sont pas suffisantes pour qu'un intrus éventuel, même disposant de moyens importants, puisse reconstituer la totalité du fichier.

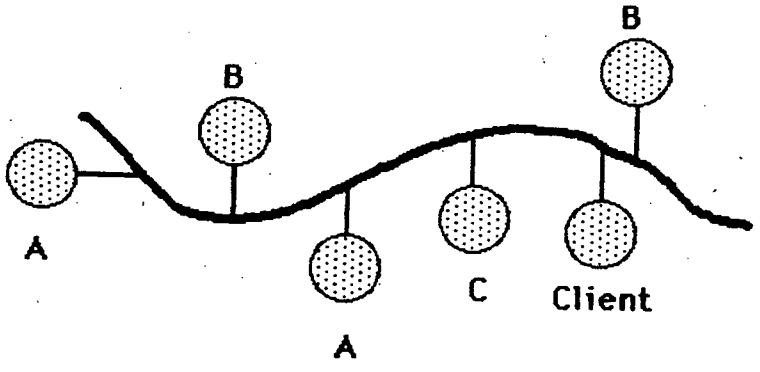


Figure 4
Fragmentation-dissémination sur réseau à bus

De même que pour la saturation, l'efficacité des mécanismes de fragmentation-dissémination est améliorée par l'utilisation de communications à diffusion.

III. OBJECTIFS DU PROJET SATURNE

Tout d'abord, précisons les limites de notre étude :

- nous ne prétendons pas concevoir de nouveaux mécanismes pour garantir la cohérence de bases de données réparties, à copies multiples, en environnement multi-accès,
- nous ne cherchons pas à définir de nouvelles méthodes d'ordonnancement de transactions, avec atomicité et cohérence [Le Lann 85],

mais nous nous proposons de développer des techniques de tolérance aux fautes susceptibles de faciliter la solution de ces problèmes : les mécanismes de masquage d'erreurs permettent de concevoir des protocoles et des algorithmes plus simples et plus efficaces pour réaliser ces fonctions.

Nous avons déja indiqué qu'un réseau à diffusion permet une mise en oeuvre plus efficace des mécanismes de saturation et de fragmentation-dissémination. Cependant, il n'est pas nécessaire qu'un tel réseau garantisse une "diffusion fiable". En effet, nous admettons comme hypothèses de fautes :

- qu'un message diffusé puisse être mal recu ou pas recu du tout par une minorité de sites destinataires,
- qu'un message exédentaire (fictif) puisse être reçu par une minorité de sites destinataires.

En fait, les fautes de transmission de ce type doivent être tolérées grâce aux mécanismes de masquage d'erreurs, aussi bien que doivent l'être les fautes de traitement, dans la mesure où seule une minorité des sites est concernée par ces fautes.(*)

Notre premier objectif est donc de fournir des services de base de traitement réparti et de transmission tolérant de larges classes de fautes, ce qui permet de simplifier la réalisation de services de plus haut niveau tels que bases de données réparties, gestion de transactions, ordonnancement de tâches temps-réel, ...

De plus, nous nous proposons de mettre au point des mécanismes permettant d'empêcher un éventuel intrus d'obtenir des informations significatives, même s'il réussit à accéder à un ou plusieurs sites.

L'interface que présente le système aux programmeurs de l'application doit être simple : l'attention du programmeur doit se porter uniquement sur son application et non sur les mécanismes de tolérance aux fautes et aux intrusions. Ainsi, la gestion des copies multiples de processus et la gestion de la fragmentation-dissémination doivent être cachées à l'application : la création/destruction de copies multiples, les votes, la dissémination des fragments de fichiers doivent être transparents, seuls doivent être visibles la création/terminaison des processus, les envois et réceptions de messages, les lectures et écritures de fichiers, ...

Comme le nombre d'exemplaires de chaque processus varie dans le temps selon la charge du système et de la criticité des processus actifs, nous sommes conduits à définir un algorithme, à la fois réparti et robuste, d'allocation des sites aux processus en fonction de critères multiples : criticités, charge du système, efficacité, spécialisation de sites, ... Pour cette étude, nous collaborons avec le projet SCORE de l'INRIA qui a déjà obtenu des résultats dans des domaines similaires [Sédillot 83].

Pour programmer les applications réparties, nous avons choisi le modèle CHORUS [Guillemont 84], projet INRIA de conception d'architecture répartie. Ceci signifie que l'interface du système de SATURNE sera probablement compatible avec celle de CHORUS. Dans ce but, nous avons établi une étroite collaboration avec l'équipe CHORUS de l'INRIA. Notre choix est motivé par les raisons suivantes :

(*) Ceci ne signifie pas que nous puissions, par ces mécanismes, tolérer toutes les fautes de transmission : certaines classes de fautes de transmission sont susceptibles de mettre en défaut notre politique de masquage d'erreurs. C'est le cas en particulier pour :

- le blocage ou la surcharge du réseau par un site défaillant,
- la partition du réseau en plusieurs sous-réseaux isolés.

Ces fautes ne peuvent être tolérées qu'à l'aide de moyens spécifiques, au niveau de l'architecture du réseau et au niveau des protocoles, tels que : double-bus, double-boucle, commutateur de contournement, ...

- CHORUS est relativement facile à implanter et à valider : il

comporte peu de primitives, il a déjà été implanté sur différentes machines, dans différents environnements, ...
- l'équipe CHORUS s'intéresse depuis longtemps à la tolérance aux fautes, et a déjà obtenu des résultats dans ce domaine [Banino 85]; cette convergence de motivation nous paraît devoir favoriser notre collaboration.

IV. LA SATURATION

Comme nous venons de l'indiquer, l'interface entre le système et l'application sera, dans SATURNE, compatible avec celle de CHORUS. Les mécanismes de saturation devront donc fournir un support adapté à cette interface. C'est pourquoi, nous allons d'abord présenter les principes de CHORUS, puis nous montrerons comment la saturation peut être mise en oeuvre de manière cohérente avec ces principes.

IV.1 Le système CHORUS

Le système CHORUS se caractérise par sa modularité, sa souplesse et sa relative indépendance vis-à-vis des machines et des réseaux de communication utilisés. L'architecture globale du système CHORUS est schématisée sur la figure 5.

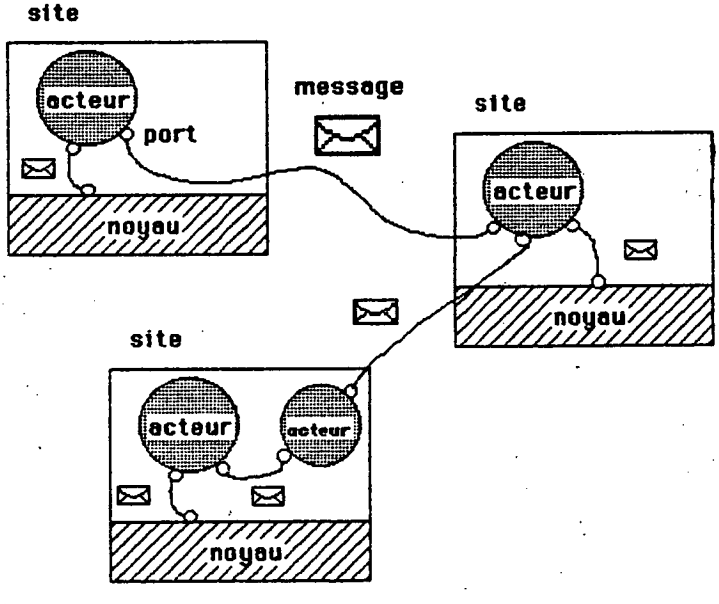


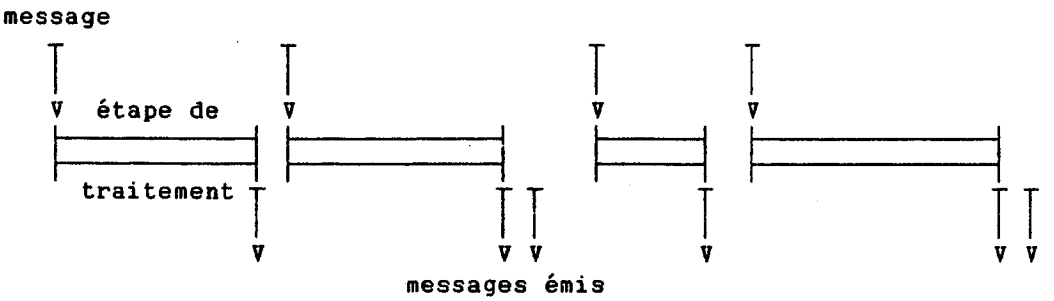
Figure 5
L'architecture CHORUS

Dans l'architecture CHORUS, le système réparti se compose d'un ensemble d'entités actives autonomes, appelées acteurs, installées sur un réseau d'ordinateurs. Un acteur peut être considéré en première

approximation, comme un processus séquentiel avec son code, ses données et son contexte.

Les acteurs sont totalement indépendants. En effet, ils s'exécutent de manière concurrente, ils ne partagent pas de données et interagissent seulement par échange asynchrone de messages. L'interface unique pour les communications est fondée sur la notion de portes logiques qui assurent une indépendance totale entre traitements et communications. Les portes sont attachées dynamiquement aux acteurs. L'échange de messages entre acteurs se fait simplement de la manière suivante : un premier acteur envoie un message à partir de l'une de ses portes vers une porte de l'acteur destinataire. Les portes sont bi-directionnelles et peuvent donc être utilisées pour émettre ou recevoir des messages. L'acteur émetteur d'un message adresse directement la porte de l'acteur destinataire sans se préoccuper de la localisation de cette porte. Les portes possèdent des noms uniques dans le système : cette désignation des portes n'est liée ni à la localisation de la porte, ni à l'acteur auquel elle est attachée. Le concept de porte permet de séparer l'organisation des communications de l'organisation des traitements, et facilite donc d'éventuelles réorganisations au niveau des acteurs.

Le schéma d'exécution des acteurs se présente sous la forme d'unités d'exécution appelées étapes de traitement. Une étape de traitement est déclenchée dans un acteur lors de la réception d'un message sur l'une de ses portes. Inversement, un acteur prépare des messages au cours d'une étapes de traitement. Ces messages ne seront émis que lorsque l'étape de traitement sera terminée. Aucun message n'est émis si l'étape de traitement ne se termine pas correctement. Une étape de traitement est l'unité élémentaire d'exécution dans une application répartie. Les étapes de traitement sont regroupées logiquement dans un acteur pour réaliser une fonction en partageant une structure de données commune. Ces étapes s'exécutent séquentiellement à l'intérieur d'un acteur. Leur influence sur l'environnement ne s'opérant que lorsque elles se terminent, elles offrent un niveau de granularité permettant de bâtir des mécanismes de tolérance aux fautes. Les acteurs, comme les portes, sont gérés dynamiquement (création, destruction, ...).



Le système exécutif est lui-même bâti sur des acteurs (les acteurs système), pour la réalisation de services spécifiques, gestion des communications par exemple. Les acteurs systèmes présents sur chaque site sont chargés de la gestion des ressources locales. Les acteurs (systèmes et d'application) s'exécutent sur un noyau de système minimal.

Cette architecture de système est très homogène, puisque l'interface entre le noyau et les acteurs (système ou d'application) est réalisée au moyen de portes. Le noyau restant simple et de petite taille permet un transport aisé sur des machines différentes. Enfin, il semble que cette architecture soit bien adaptée à la description et à la programmation d'applications réparties, comme l'ont montré les expérimentations menées par l'équipe CHORUS et les utilisateurs de ce système.

IV.2 Allocation des sites pour la saturation

Dans le système SATURNE, plusieurs copies d'un acteur exécutent la même étape de traitement sur différents sites; les messages d'information sont votés, à la terminaison de l'étape de traitement, suivant un schéma de vote à seuil.

Comme nous l'avons décrit au paragraphe précédent, les acteurs se composent de plusieurs étapes de traitement qui peuvent partager des données rémanentes. La mise à jour de ces zones de données doit entraîner la même mise à jour sur tous les exemplaires des acteurs, de manière à provoquer une exécution identique de la prochaine étape de traitement, quels que soient les exemplaires de l'acteur qui seront impliqués dans ce traitement. Cette mise à niveau des données rémanentes est réalisée au moyen de messages de contrôle spécifiques, transmis à la fin de l'étape de traitement, et sur lesquels un vote sera appliqué de la même manière que sur les messages d'information.

L'allocation des sites, pour l'exécution des étapes de traitement, doit obéir à certaines règles :

- autant que faire se peut, à tout instant, tous les sites doivent exécuter une étape de traitement (saturation), de telle sorte que le nombre d'exemplaires d'une séquence active puisse varier durant son exécution, mais soit aussi grand que possible (redondance maximale),

- nous appellerons "criticité" le nombre minimum d'exemplaires de chaque étape de traitement. La criticité est un paramètre d'initialisation de l'étape de traitement : une étape de traitement ne peut être lancée que si le nombre de ses exemplaires est plus grand ou égal à sa criticité,

- la charge doit être équilibrée de manière à ce que le nombre d'exemplaires de chaque étape de traitement active soit proportionnel à sa criticité,

- la latence d'erreur sera minimisée par des mécanismes de détection et de diagnostic (détection de la divergence des résultats par les voteurs, programmes d'autotest tournant sur les sites oisifs, rapports d'erreurs sur des sites observateurs [Ayache 82], ...)

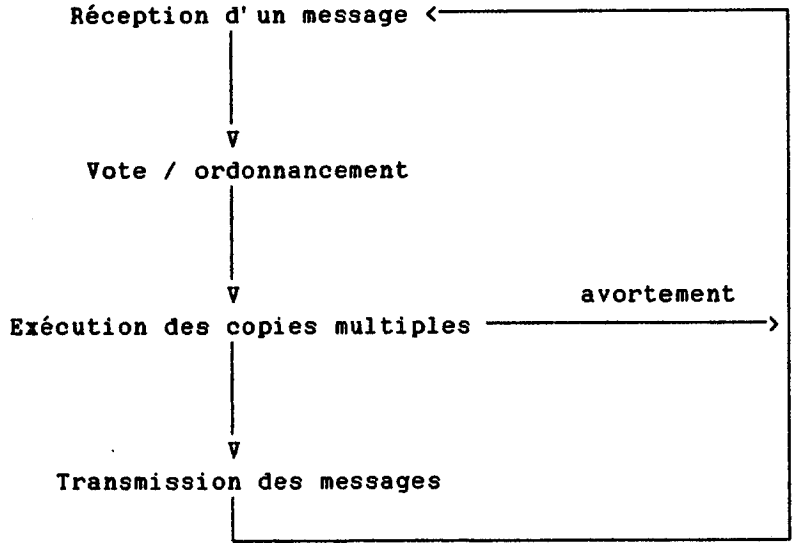
- le principe d'exclusion entre les différentes étapes de traitement d'un acteur doit être respecté : si une étape de traitement est en cours sur certains exemplaires d'un acteur,

aucune autre étape de traitement ne peut être déclenchée sur cet acteur (même sur d'autres exemplaires),

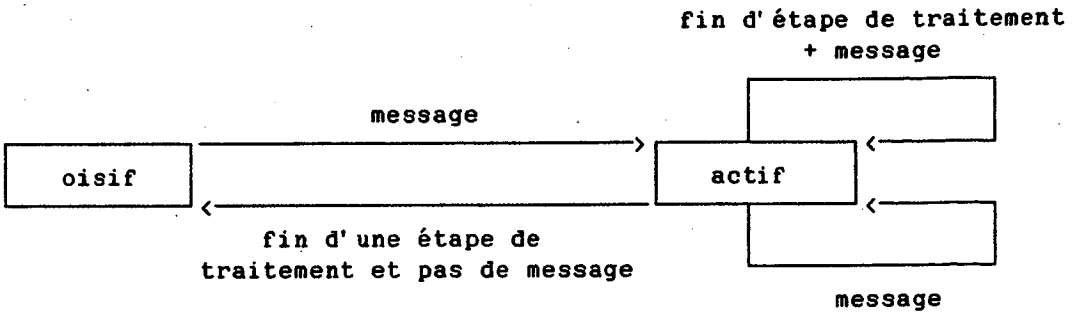
- nous appellerons ordonnanceur le logiciel chargé de l'allocation des sites; ce logiciel doit être réparti sur tous les sites du réseau, pour éviter un "point dur" dans le système; les exécutions des différents ordonnanceurs doivent être cohérentes; cette cohérence est maintenue au moyen de vote sur les données locales de chaque ordonnanceur.

Les sites du réseau sont divisés en plusieurs classes : les sites utilisateurs, les sites d'archivage, les sites de sécurité, les sites de traitement spécialisés, ... Tout site d'une classe possède un exemplaire de chaque acteur de la classe. Lorsqu'un message à destination d'un acteur d'une certaine classe est transmis sur le réseau à diffusion, l'unité réceptrice des sites récupère toutes les copies de ce message (messages identiques émis par les différents exemplaires actifs des acteurs). Lorsque le nombre de messages identiques dépasse le seuil, l'ordonnanceur local est activé pour décider si le site doit exécuter l'étape de traitement correspondante ou non. Cette décision dépend de l'activité du site, de celle des autres sites de la classe, de la criticité du nouveau message à traiter et des criticités des étapes de traitement en cours.

Les différentes phases de l'exécution d'un traitement sont les suivantes:



Les deux états possibles d'un site sont l'état "oisif" et l'état "actif". Au moment de la connexion d'un site sur le réseau, le site est dans l'état "oisif". Lorsqu'un message destiné à un acteur de la classe du site en question est transmis sur le réseau, et lorsque un nombre suffisant de copies identiques de ce message sont reçues (principe du vote à seuil), le site devient "actif" et exécute l'étape de traitement correspondante.



Lorsqu'un nouveau message destiné à un autre acteur de la même classe est reçu, l'ordonnanceur local doit décider, soit d'abandonner l'étape de traitement en cours pour exécuter la nouvelle étape, soit de poursuivre l'exécution de l'étape de traitement courante (cette décision dépendant des règles que nous avons énoncées précédemment). Lorsqu'une étape de traitement se termine, s'il existe une étape de traitement en attente ou si l'étape de traitement qui vient de se terminer a émis un message vers un acteur de la même classe, le site reste actif et exécute la nouvelle étape de traitement. Dans le cas contraire, s'il n'existe pas d'étape de traitement candidate, le site devient "oisif".

Tout site dans l'état "oisif" exécute des programmes d'autotest pour détecter des erreurs latentes éventuelles que les programmes d'application ont une faible probabilité de détecter : dernières adresses en mémoire principale ou du disque, instructions peu fréquemment utilisées, ...

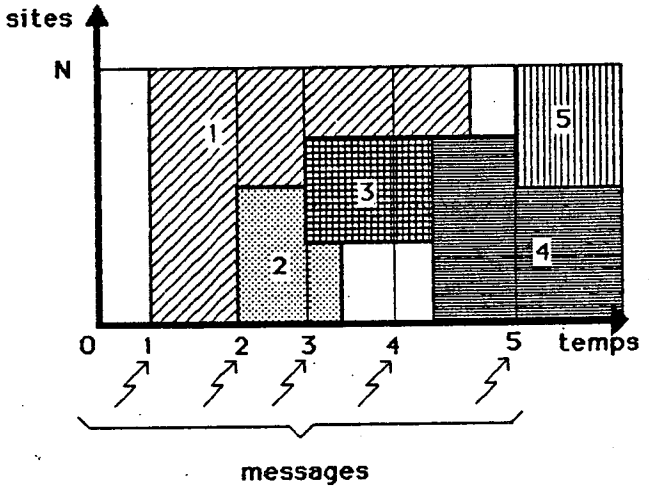


Figure 6
Ordonnancement par saturation (exemple)

Pour décrire plus simplement les règles d'ordonnancement, nous allons prendre un exemple (figure 6). Supposons qu'à l'instant 0, tous les N sites soient "oisifs" (nous ne considèrerons ici que le cas d'une seule classe). A l'instant 1, un message est pris en compte (vote), avec une criticté légèrement inférieure à N/3. Tous les sites deviennent alors "actifs" et exécutent l'étape de traitement 1 correspondante. A l'instant 2, un

message 2 ayant la même cricité que le message 1 arrive. La moitié des sites exécutant le processus 1 doivent alors interrompre leur traitement pour exécuter le processus 2 (équilibrage de charge). A l'instant 3, un troisième message arrive avec la même criticité que les deux messages précédents. D'où, N/3 sites doivent exécuter chacune des trois étapes de traitement. Supposons qu'alors le processus 2 se termine sans émettre de message. Tant qu'il n'y a pas de nouveau message, les sites exécutant le processus 2 deviennent "oisifs". A l'instant 4, un nouveau message arrive avec une criticité supérieure à N/3, et telle que :

criticité (message 1) + criticité (message 3) + criticité (message 4) > N

Il n'est donc pas possible d'exécuter simultanément les trois étapes de traitement correspondantes. L'exécution de l'étape 4 sera retardée jusqu'à ce qu'il y ait suffisamment de ressources en terme de sites. Ce qui arrive lorsque le processus 3 se termine sans émettre de nouveau message. Tous les sites "oisifs" ainsi que ceux qui exécutaient le processus 3, vont alors exécuter l'étape de traitement 4. Lorsque le processus 1 se termine (sans émettre de message), tous les sites correspondant deviennent "oisifs". A la réception du message 5 dont la criticité est approximativement égale à celle du message 4, et telle que :

criticité (message 4) + criticité (message 5) ≤ N

tous les sites "oisifs", ainsi qu'une partie des sites exécutant le processus 4 vont exécuter le processus 5.

Dans cet exemple, on peut constater que l'équilibre de charge n'est pas toujours parfaitement réalisé (par exemple lors de l'exécution concurrente des processus 1 et 4). Ceci est dû au fait que les sites "oisifs" ne peuvent exécuter que des étapes de traitement sur de nouveaux messages (il n'est pas possible de prendre une étape de traitement "en cours de route"), et que le principe de "saturation" (tout site doit être aussi "actif" que possible) est plus important que le principe "d'équilibre de charge".

Dans le but de maintenir la cohérence globale des décisions des ordonnanceurs locaux, les données rémanentes des ordonnanceurs sont votées à chaque instant de décision : initialisation d'une étape de traitement, fin d'une étape de traitement, mise en file des messages ...

V. LA FRAGMENTATION-DISSÉMINATION

V.1 Le service d'archivage

Dans le domaine de la tolérance aux intrusions, nous nous proposons de focaliser notre attention sur un service d'archivage. Ce type de service est particulièrement utile dans les systèmes répartis, qu'il s'agisse de réseaux locaux ou de réseaux à plus grande échelle : un utilisateur souhaite pouvoir récupérer les informations qu'il aura archivées, quelle que soit le site sur lequel il se connecte. D'autre part, le service

d'archivage est particulièrement sensible aux intrusions : il est relativement facile, pour un intrus, d'obtenir beaucoup d'information en violant les accès à un site d'archivage, ou en volant un support de mémorisation (disque, bande magnétique, ...); par contre, il est relativement difficile de s'introduire sur la station de travail d'un utilisateur pendant sa connexion, et les informations ainsi obtenues sont beaucoup plus limitées.

Pour des raisons de simplicité, nous nous placerons, pour ce chapitre, dans l'hypothèse d'un réseau local constitué par (figure 7) :

- des sites utilisateurs, composés de stations de travail (ou ordinateurs personnels) ayant des capacités de mémorisation (disques) et de traitement locales, non partageables (le site utilisateur est exclusivement réservé à un utilisateur pendant toute sa période de connexion),

- des sites d'archivage, chargés de mémoriser toutes les informations des utilisateurs en dehors de leurs périodes de connexion,

- des sites de sécurité, dont nous verrons le rôle plus loin,

- un support de communication à diffusion.

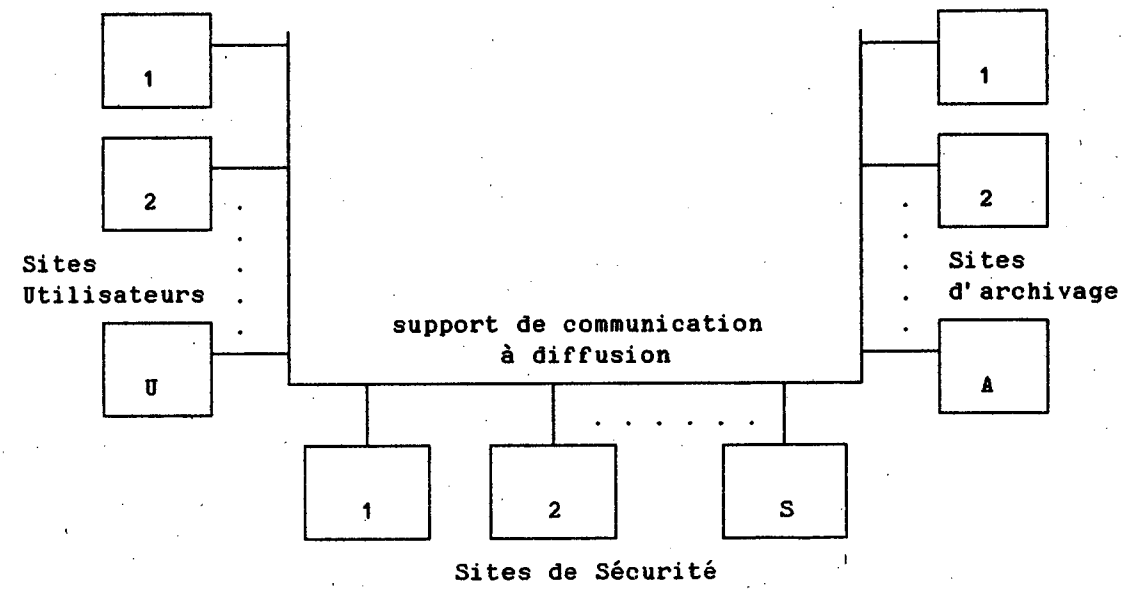


Figure 7
Structure du système d'archivage

Le fonctionnement du système d'archivage est schématiquement le suivant :

Lors de sa connexion sur un site utilisateur quelconque, l'utilisateur peut transférer (globalement) des sites d'archivage vers son site local certains des fichiers qu'il a précedemment archivés. Il peut alors modifier localement ces fichiers et en

créer d'autres. Avant sa déconnexion, il peut archiver ces fichiers par transfert vers les sites d'archivage, puis détruire les copies locales de façon à les rendre inaccessibles pour le prochain utilisateur se connectant sur le même site. De même, pendant sa connexion, l'utilisateur peut détruire certains de ses fichiers archivés.

Les services d'archivage fournis aux utilisateurs sont donc :
- la création et l'écriture d'un fichier archivé,
- la lecture d'un fichier archivé,
- la destruction d'un fichier archivé.

Pour vérifier si un utilisateur peut ou non mettre en oeuvre l'un de ces services sur un fichier archivé, le système d'archivage gère des droits d'accès d'une manière analogue à celle d'UNIX :

Les fichiers possèdent un nom unique correspondant à leur "chemin d'accès" dans une arborescence de répertoires. A tout fichier est associée une liste de droits d'accès. Les droits sont de trois types : les droits de lecture, d'écriture (modification, destruction) et le droit d'exécution. Les droits d'accès pour chaque fichier sont indiqués pour trois classes distinctes :
- le propriétaire (l'utilisateur créateur du fichier),
- le groupe d'utilisateurs auquel appartient le propriétaire,
- les autres utilisateurs.

Les droits d'accés au fichier de chaque classe sont établis (modifiés) par le propriétaire du fichier.

D'autre part, pour garantir la cohérence des fichiers archivés, un service d'exclusion classique est fourni sur les périodes critiques comprises entre l'ouverture et la fermeture des fichiers, avec :
- possibilité de multiples lectures simultanées d'un fichier,
- exclusion de l'écriture par rapport aux lectures et aux autres écritures.

Notre objectif est de fournir ces services généraux à l'utilisateur, tout en tolérant un certain nombre de fautes (par exemple l'indisponibilité d'un ou plusieurs sites d'archivage et/ou de sécurité) et d'intrusions (par exemple la violation des accès à un ou plusieurs sites d'archivage et/ou de sécurité).

V.2 Sites de sécurité

Les sites de sécurité sont responsables de la gestion des droits d'accès et des exclusions : ils remplissent donc un service analogue à celui du gestionnaire de répertoire sous UNIX. Ils doivent conserver les verrous associés à chaque fichier pour satisfaire les règles d'exclusion citées plus haut. Ils doivent conserver pour chaque fichier une "clé de fragmentation", nécessaire au réassemblage du fichier archivé. Ils doivent, de plus, conserver pour chaque utilisateur une "clé d'authentification", qui permette de reconnaître les droits de chaque utilisateur sur chaque fichier. L'ouverture d'un fichier archivé par un utilisateur autorisé correspond à l'émission par les sites d'archivage de la clé de fragmentation du fichier et d'un "activateur" d'ouverture vers le site

utilisateur, et à l'émission vers les sites d'archivage de l'activateur d'ouverture [Fraga 85].

Les informations que gèrent les sites de sécurité doivent être suffisamment redondantes afin que l'indisponibilité d'un ou de quelques sites de sécurité ne provoque pas l'indisponibilité du service d'archivage. Par ailleurs, ces informations doivent être suffisamment protégées pour qu'un intrus qui réussirait à voler ou à modifier les informations d'un ou de quelques sites de sécurité, ne puisse pas pour autant obtenir des accès non autorisés aux fichiers archivés. Ces deux objectifs peuvent paraître contradictoires : le premier doit faciliter l'accès aux informations, même en cas de faute; le deuxième doit en rendre l'accès aussi difficile que possible.

Le problème pourrait être résolu par des méthodes cryptographiques classiques : tous les sites de sécurité auraient les mêmes informations, mais cryptées avec des clés différentes; il suffit alors qu'un des sites de sécurité autorise la demande d'ouverture pour que le site utilisateur puisse effectuer ses transferts. Mais cette méthode ne présente pas des caractéristiques de confidentialité réellement satisfaisantes : un intrus qui réussirait à voler les informations d'un seul site de sécurité, même cryptées, pourrait, avec suffisamment de puissance de calcul et de temps, obtenir les informations nécessaires pour pouvoir accéder à tous les fichiers.

Il existe une autre solution plus satisfaisante et plus cohérente avec le principe de la fragmentation [Fraga 85]. Cette solution consiste à cacher les informations des sites de sécurité par un "schéma à seuil" ([Shamir 79], [Davida 80]). Ceci signifie que les informations d'un seul site de sécurité ne sont pas suffisantes pour ouvrir un fichier : il faut qu'une majorité des sites de sécurité soient d'accord, et il faut que le site utilisateur, comme les sites d'archivage, combine les informations reçues d'au moins la moitié des sites de sécurité pour pouvoir accéder à un fichier. Ainsi, par exemple, les clés de fragmentation ne sont pas directement mémorisées par les sites de sécurité, mais chaque site de sécurité garde sa propre "image" de chaque clé; pour reconstituer cette clé, le site utilisateur doit récuperer l'image de la clé d'au moins la moitié des sites de sécurité : on tolère ainsi l'indisponibilité ou l'intrusion de la moitié des sites de sécurité moins un.

V.3 Fragmentation et dissémination d'un fichier

La fragmentation-dissémination consiste à découper le fichier en fragments selon un algorithme dépendant de la clé de fragmentation, de sorte qu'aucun fragment ne contienne suffisamment d'information pour être significatif, puis à disséminer ces fragments parmi les sites d'archivage.

La fragmentation est à la charge du site utilisateur qui a réussi à reconstituer la clé de fragmentation d'après les images de clés transmises par les sites de sécurité. La dissémination est mise en oeuvre par les sites d'archivage : lors de la demande d'écriture d'un fragment, diffusée sur le réseau par le site utilisateur, chaque site d'archivage doit décider d'archiver ou non le fragment, en fonction du nombre de copies souhaitées (redondance pour tolérer les fautes des sites d'archivage), et en fonction des décisions que doivent prendre les autres sites d'archivage. Cette

décision est à la charge d'un algorithme analogue à celui des ordonnanceurs utilisés par la saturation. Sur les sites d'archivage, rien ne permet de reconnaître les fragments qui appartiennent à un même fichier : chaque fragment porte un nom indépendant formé par le site utilisateur à partir de la clé de fragmentation du fichier; seul le site qui connaît cette clé peut donc regénérer les noms des fragments du fichier.

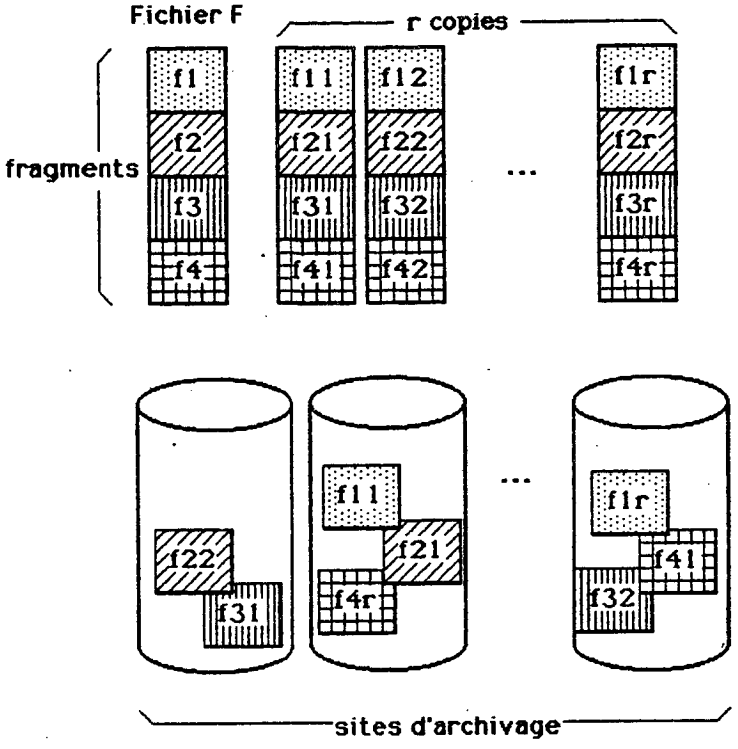


Figure 8
Fragmentation-dissémination

L'écriture d'un fichier sur le système d'archivage se fait de la manière suivante :

- le site utilisateur émet une demande d'ouverture en écriture du fichier vers les sites de sécurité,
- s'il reconnaît l'utilisateur (authentification), si cet utilisateur a le droit en écriture sur le fichier, et si le fichier n'est pas ouvert, chaque site de sécurité émet vers le site utilisateur son image de la clé de fragmentation du fichier et un activateur associé à l'ouverture, et émet vers les sites d'archivage ce même activateur,
- le site utilisateur combine les images de clé qu'il a reçues pour reconstituer la clé de fragmentation,
- le site utilisateur et les sites d'archivage combinent les activateurs transmis par les sites de sécurité pour reconstituer un activateur global (schéma à seuil),
- le site utilisateur utilise la clé de fragmentation pour fragmenter le fichier et générer les noms de chaque fragment,
- pour chaque fragment, le site utilisateur emet une demande d'écriture du fragment (avec son nom et avec l'activateur) sur le réseau à diffusion à destination des sites d'archivage,
- chaque site d'archivage, s'il reconnaît l'activateur, décide

d'enregistrer ou non le fragment,
- lorsque tous les fragments ont été émis, le site utilisateur émet une demande de fermeture du fichier vers les sites de sécurité,
- les sites de sécurité émettent chacun un "désactivateur" vers les sites d'archivage,
- chaque site d'archivage combine les désactivateurs reçus des sites de sécurité pour reconnaître l'activateur concerné, et détruisent localement cet activateur.

La lecture du fichier se fait de la manière suivante :
- le site utilisateur émet une demande d'ouverture en lecture du fichier vers les sites de sécurité,
- s'il reconnaît l'utilisateur (authentification), si cet utilisateur a le droit en lecture sur le fichier, et si le fichier n'est pas ouvert en écriture, chaque site de sécurité émet vers le site utilisateur son image de la clé de fragmentation du fichier et un activateur associé à l'ouverture, et émet vers les sites d'archivage ce même activateur,
- le site utilisateur combine les images de clé qu'il a reçues pour reconstituer la clé de fragmentation,
- le site utilisateur et les sites d'archivage combinent les activateurs transmis par les sites de sécurité pour reconstituer un activateur global (schéma à seuil),
- le site utilisateur utilise la clé de fragmentation pour générer les noms de chaque fragment,
- pour chaque fragment, le site utilisateur emet une demande de lecture du fragment (avec son nom et avec l'activateur) sur le réseau à diffusion à destination des sites d'archivage,
- chaque site d'archivage, s'il reconnaît l'activateur, recherche le fragment, et s'il le trouve, émet le fragment,
- lorsque tous les fragments ont été émis, le site utilisateur réassemble le fichier à l'aide de la clé de fragmentation, puis émet une demande de fermeture du fichier vers les sites de sécurité,
- les sites de sécurité émettent chacun un désactivateur vers les sites d'archivage,
- chaque site d'archivage combine les désactivateurs reçus des sites de sécurité pour reconnaître l'activateur concerné, et détruisent localement cet activateur.

Il faut souligner que ces schémas de lecture et d'écriture de fichier sont en pratique encore compliqués par le fait que :
- les demandes de lecture et d'écriture sont en fait exécutées par des processus redondants (saturation), et que, par conséquent, les messages sont émis en plusieurs exemplaires qu'il convient de voter selon les algorithmes de vote majoritaire de la saturation,
- les messages de service doivent être cryptés de sorte qu'un intrus ne puisse les falsifier (problème d'authentification),
- en lecture, il n'est pas nécessaire que tous les sites d'archivage émettent leur propre exemplaire de chaque fragment, il est possible de vérifier la cohérence d'un fragment par une somme de contrôle [Fraga 85], de façon à réduire le nombre de messages émis sur le réseau.

CONCLUSION

Nous n'avons jusqu'à présent défini que les principes de la saturation et de la fragmentation-dissémination. Ce sont deux techniques originales qui nous semblent suffisamment prometteuses pour mériter une expérimentation qui permette à la fois de vérifier leur faisabilité et d'évaluer les coûts de leur mise en oeuvre.

Il faudra donc développer, valider et exploiter en vraie grandeur sur un réseau de stations de travail, des logiciels de gestion de la saturation et de la fragmentation-dissémination. Ces logiciels, répartis, doivent être robustes et souples pour permettre d'évaluer différentes politiques d'allocation de sites, de vote, de fractionnement, de dissémination, etc... De plus, ces logiciels se caractérisent par un fonctionnement autonome de chaque site avec une connaissance imprécise du comportement des autres sites. L'ensemble de ces caractéristiques peut nous conduire à concevoir des méthodes et outils de développement et de vérification originaux, qu'il s'agisse de moyens de vérification formelle, de simulation ou d'évaluation prévisionnelle de performances et de sûreté de fonctionnement. Soulignons que la vérification ne doit pas se limiter aux aspects fonctionnels du système, mais doit prendre également en compte le comportement du système en présence de fautes et d'intrusions.

La réalisation d'un système expérimental sur un réseau de stations de travail doit permettre des mesures précises des performances et des surcoûts, aussi bien pour ce qui concerne la puissance de traitement des stations de travail que le débit du réseau, en fonctionnement idéal comme en présence de fautes, et en faisant varier différents paramètres tels que la charge du système, l'interactivité des processus, la taille des fichiers, la concurrence des accès aux fichiers, le nombre de sites, etc... Pour effectuer ces mesures, il pourra être nécessaire d'utiliser des outils d'injection d'erreurs du type de MESSALINE [Arlat 84].

L'évaluation de la confidentialité ouvre un véritable champ d'investigation, aussi bien pour la définition d'un ensemble de mesures adaptées que pour la comparaison de différentes techniques.

RÉFÉRENCES

Arlat 84

J. ARLAT, J. P. BLANQUART, J. C. LAPRIE: "Sur la certification des systèmes informatiques : le projet EVE - Application au Poste d'Aiguillage Informatisé", Actes du 4ème Colloque Int. sur la Fiabilité et la Maintenabilité, Perros-Guirec, mai 1984, pp. 650-656

Avizienis 85

A. AVIZIENIS, P. GUNNINGBERG, J. P. J. KELLY, L. STRIGINI, P. J. TRAVERSE, K. S. TSO, U. VOGES: "The UCLA DEDIX system: a distributed testbed for multiple-version software", Proc. 15th Int. Symp. on Fault-tolerant Computing (FTCS-15), Ann Arbor, June 1985, pp. 126-134

Ayache 82

J. M. AYACHE, J. P. COURTIAT, M. DIAZ: "Self-checking software in distributed systems", Proc. of the 3rd International Conference on Distributed Computing Systems, Miami, October 1982, pp. 163-170

Banino 85

J. S. BANINO, J. C. FABRE, M. GUILLEMONT, G. MORISSET, M. ROZIER: "Some fault-tolerance aspects in the CHORUS distributed system", Proc. 5th Int. Conf. on Distr. Computing Systems, Denver, May 1985, pp. 430-437

Castillo 81

X. CASTILLO, D. P. SIEWIOREK: "Workload, performance, and reliability of digital computing systems", Proc. of the 11th International Symposium on Fault-tolerant Computing (FTCS-11), June 1981, pp. 84-89

Davida 80

G. I. DAVIDA, R. A. DEMILLO, R. J. LIPTON: "Protecting shared cryptographic keys", 1980 Symp. on Security and Privacy, Oakland, April 1980, pp. 100-102

Fraga 85

J. FRAGA: "Sécurité des données par la tolérance des intrusions", Thèse de Doctorat, INP-LAAS, Toulouse, 12 juillet 1985, 157 pages

Frost 85

FROST & SULLIVAN, Etude de marché rapportée dans EDN, February 7, 1985, p. 344

Guillemont 84

M. GUILLEMONT, H. ZIMMERMANN, G. MORISSET, J. S. BANINO: "CHORUS: une architecture pour les systèmes répartis", Rapport de recherche INRIA n°274, Mars 1984, 77 pages.

Iyer 82

R. K. IYER, S. E. BUTNER, E. J. McCLUSKEY: "A statistical failure/workload relationship: results of a multi-computer study", IEEE Transactions on Computers, Vol. C-31, N°7, July 1982, pp. 697-705

Koga 82

Y. KOGA, E. FUKUSHIMA, K. YOSHIHARA: "Error recoverable and securable data communication for computer network", Proc. 12th Int. Symp. on Fault-tolerant Computing (FTCS-12), June 1982, pp. 183-186

Laprie 84
J. C. LAPRIE: "Sûreté de fonctionnement des systèmes informatiques et tolérance aux fautes : concepts de base", Rapport de recherche LAAS n°84.051, juillet 1984, 30 pages

Le Lann 85
G. LE LANN: "Distributed real-time processing", Internationnal Symposium on Computing Systems for Process Control (Brown Boveri), Zurick, September 1985, 29 pages

Sédillot 83
S. SEDILLOT: "Flexible scheduling techniques for a multiple robot system", Symposium on Production Management, Brussels (Belgium), March 1983, 17 pages

Shamir 79
A. SHAMIR: "How to share a secret", Communications of the ACM, Vol. 22, n°11, November 1979, pp. 612-613

Shoch 82
J. F. SHOCH, J. A. HUPP: "The "WORM" programs - Early experience with distributed computation", Communications of the ACM, March 1982, Vol. 25, N°3, pp. 172-180

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique