quement par des séquences d'événements improbables. Lorsqu'un système utilise la duplication active, les différentes copies voient les mêmes mises à jour dans le même ordre ; toutefois, les mises à jour sont seulement une petite partie des requêtes reçues par un programme. La plupart du temps, les programmes dupliqués travaillent en parallèle, chaque programme traitant son propre ensemble de requêtes dans un ordre spécifique. Dès lors, même si une erreur logicielle échappait aux tests et interférait avec un membre d'un logiciel réparti, il serait improbable qu'elle mette hors service tous les membres d'un groupe de traitement en même temps.

L'idée de la duplication active est simple, mais sa mise en œuvre est difficile. La gestion dynamique des listes de participants, qui changent continuellement, et les communications avec les groupes de processus sont sujettes aux erreurs systèmes et à la perte de messages. Bien que l'informatique répartie se soit imposée au cours de la dernière décennie, la duplication active n'est sortie que récemment des laboratoires de recherche.

La boîte à outils des réseaux robustes

Durant ces dernières années, plus d'une dizaine d'équipes d'ingénieurs ont mis au point des logiciels qui assurent la robustesse des systèmes répartis. Tous offrent une haute disponibilité grâce à la duplication active, mais ils diffèrent sur leurs priorités : les uns privilégient la rapidité ; d'autres, la sécurité.

À l'Université Cornell, nous avons réalisé deux logiciels. L'un d'entre nous (K. Birman) a dirigé l'équipe qui a produit le logiciel *Isis* en 1987 ; puis, ensemble, nous avons construit le logiciel *Horus*, achevé en 1994. Les noms de ces programmes indiquent leurs propriétés : la déesse égyptienne Isis ressuscita le dieu Osiris après qu'il avait été tué, lors d'un duel, par Seth, dieu de la guerre ; Horus, fils d'Isis, vainquit finalement Seth. De même, les logiciels *Isis* et *Horus* établissent le fonctionnement d'un système distribué interrompu par une défaillance.

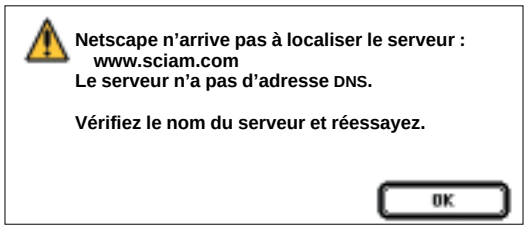
Des logiciels comme *Isis* utilisent un assortiment de fonctions logicielles, ou «outils», qui dupliquent et mettent à jour les données, gardent la trace des groupes de processus et aident à gérer l'appartenance de ces groupes. *Isis* peut aussi répartir le traitement des données entre les serveurs (une procédure nommée partage des tâches). Les systèmes répartis qui utilisent le partage des tâches présentent la plupart des avantages du parallélisme sans recourir à des ordinateurs parallèles spécialisés. En divisant le travail entre de nombreux serveurs fonctionnant de concert, *Isis* permet aux systèmes d'effectuer rapidement de lourdes tâches. De même, lorsqu'une application particulière nécessite davantage de puissance de calcul, on peut ajouter un serveur, en conservant la technique de partage des tâches pour le nouveau groupe. Les performances d'*Isis* et d'*Horus* surprennent souvent les programmeurs, qui considèrent qu'un système robuste est nécessairement lent et onéreux.

La duplication active a été utilisée pour de nombreuses applications : télécommunications, marchés financiers, banques... En Norvège, des informaticiens ont mis au point un système de surveillance de l'environnement

Le *Web* s'emmêle

La majeure partie du *World Wide Web* est invisible. De nombreux utilisateurs ne voient qu'un programme de navigation et des serveurs distants, où les documents à charger sont stockés. Cependant les sites du réseau et le réseau lui-même, qui relie ces sites, est composé de millions d'autres programmes et de serveurs ; des dizaines d'entre eux coopèrent pour fournir un seul document. Par exemple, quand on cherche une information dans un serveur du centre de coordination européen du réseau à l'Institut national de recherche en informatique et en automatique, le nom «www.inria.fr» doit être converti en une adresse numérique que les logiciels reconnaissent. Cette tâche est effectuée par une série de programmes de conversion d'adresses avant que l'adresse correcte soit localisée. En outre, toute requête d'accès à un site du réseau passe normalement par un certain nombre de relais, qui sauvegardent une copie des documents fréquemment demandés, afin de diminuer la charge de travail des serveurs *Web* distants. Ainsi, quand un intermédiaire proche a stocké une copie du document recherché, l'utilisateur évite un long transfert de fichier à travers le réseau.

Ce système de relais est classique dans les réseaux, mais c'est une source de défaillances : quand un serveur de nom ne répond pas, ou quand un relais échoue, ou quand le serveur *Web* tombe en panne, la requête initiale n'aboutit pas. Aussi l'incontournable message d'erreur du *Web* qui signale qu'un serveur est bloqué ou occupé *(ci-dessous)* peut-il être trompeur : la surcharge ou la panne d'un nombre quelconque de programmes intermédiaires peut aussi conduire à ce message.



Le réseau *Internet* dans sa globalité peut connaître des pannes partielles assez semblables à celles des réseaux téléphonique et électrique. Ainsi, fin 1995, l'un des principaux serveurs de temps du réseau *Internet*, à Atlanta, fut surchargé de façon intermittente. Pendant ce temps, personne sur le *Web* ne pouvait obtenir de documents de serveurs dont l'adresse n'était pas déjà connue du système local. Ce type de pannes gêne un grand nombre de personnes dans le monde entier. Même si une connexion avec un serveur réussit, cela n'exclut pas des erreurs ultérieures. Les copies faites par les programmes intermédiaires du *Web* ne sont pas actualisées lorsque le document original l'est. Aussi les utilisateurs n'ont-ils aucune garantie que la version qu'ils consultent est la dernière mise à jour. Si les documents nécessaires ne sont pas gardés à jour, certaines applications dépendantes du temps perdent alors leur fiabilité. D'un côté, les programmes mandataires du *Web* améliorent la fiabilité du réseau *Internet* en diminuant sa charge de travail, mais, de l'autre, ils diminuent la fiabilité en créant l'éventualité de l'obtention d'informations périmées.

Les projets vitaux, sur le réseau *Internet* ou sur d'autres systèmes informatiques répartis, requerront la garantie que les informations extraites du système soient précises, mises à jour et toujours disponibles. On peut éviter des problèmes en sauvegardant les informations vitales par le biais de la duplication active décrite dans l'article.

fondé sur cette technique *(voir la figure 2)*. L'organisme français de contrôle du trafic aérien expérimente également cette technique pour l'appliquer à une nouvelle génération de logiciels de contrôle du trafic. Des usines de production ont utilisé les groupes de processus pour coordonner les tâches et pour reconfigurer les chaînes d'assemblage lorsque certains équipements de ces chaînes sont retirés pour être révisés.

Toutefois, à mesure que la complexité des applications augmente, les informaticiens découvrent que la duplication active a des limitations. Le partage des tâches n'est ni toujours possible, ni désirable : certains systèmes (notamment ceux où les données stockées dans un serveur changent très rapidement) ralentissent quand leurs éléments sont dupliqués. Par exemple, pour les vidéoconférences, la duplication active améliore la tolérance aux pannes du réseau de serveurs qui doivent continuer de fonctionner même quand certains participants sont déconnectés. En revanche, appliquée à la transmission de données vidéo à distance, cette technique ralentira le système sans améliorer sa fiabilité.

Le besoin de flexibilité nous a conduits à mettre au point *Horus*. *Horus* utilise à la fois la technique de duplication active et un traitement modulaire des tâches : un peu comme avec des *Lego*, les différentes pièces de construction, ou modules, d'*Horus* peuvent s'assembler selon n'importe quelle combinaison pour répondre aux besoins spécifiques d'un groupe de processus particulier. Un module chiffre les données, afin de fermer le réseau aux pirates ; un autre module identifie les erreurs de communication potentielles pouvant survenir lorsque des messages sont perdus ou altérés. Les programmeurs qui utilisent *Horus* décident des propriétés nécessaires à leur système et adaptent le système à l'usage projeté. En outre, *Horus* est extensible à l'aide de modules faits sur mesure pour des besoins spécifiques que nous n'avons pas rencontrés ni imaginés.

Horus a un nombre croissant d'utilisateurs dans le monde. À l'Université Cornell, Brian Smith l'a utilisé pour construire un système de vidéoconférences associé à des groupes de discussion et à des outils de travail coopératif assisté par ordinateur.

3. CES PASSAGERS, qui attendent leurs avions à l'aéroport de La Guardia, à New York, sont victimes d'une panne du système local de contrôle de trafic aérien faisant suite à une coupure de courant, en mai 1995. Les efforts pour rajeunir les logiciels américains vieillissants de contrôle de trafic aérien utilisant des systèmes répartis remontent aux années 1980.

Un manque de volonté?

Notre travail sur *Isis* et sur *Horus* nous a convaincus qu'une étude soignée assure la fiabilité d'un réseau informatique. Toutefois, la réalisation d'autoroutes de l'information robustes coûtera plus cher et prendra plus de temps que ne le voudraient les fabricants et les utilisateurs. Les logiciels de gestion des systèmes répartis sont habituellement construits avec la technique existante, qui n'a pas été conçue pour être fiable. En outre, les informaticiens doivent trouver des techniques nouvelles pour les très gros réseaux : un système extrêmement robuste lorsque 50 utilisateurs y accèdent simultanément peut se révéler trop lent, et donc non fiable, lorsque 5 000 personnes s'y connectent.

Bien que les programmeurs aient appliqué la technique de l'informatique répartie robuste avec succès dans certains cas, le public entend plus souvent parler des défaillances des systèmes non robustes. Ainsi, ces dernières années, des dizaines de rapports ont fait état de problèmes du système de contrôle du trafic aérien américain. Fin 1995, le système de Los Angeles tomba en panne, mettant les contrôleurs dans l'impossibilité de communiquer avec les avions ; une collision en plein ciel fut évitée de justesse.

Pis encore, les mises à jour des logiciels de contrôle de trafic aérien commandées, en 1982, par le ministère américain de l'aviation ont été retardées et les logiciels finalement proposés sont moins performants que ceux qui avait été initialement choisis : les aspects «haute disponibilité» et «distributivité» ont été presque éliminés. Pourtant, les contrôleurs aériens considèrent que le système existant est inapproprié, notamment parce qu'il lui manque une architecture de logiciel distribué et qu'il a perdu de sa fiabilité avec l'âge.

L'industrie logicielle est-elle en crise? Si une crise existe, c'est peut-être autant par manque de volonté que par manque de moyens. Tous les ingénieurs n'ont pas besoin de rendre robustes leurs logiciels réseaux : la pression du public pour accroître la fiabilité ne semble pas s'étendre au-delà de quelques applications particulièrement sensibles. Les sociétés qui commercialisent les logiciels d'informatique répartie précisent souvent, dans les garanties de leurs produits, que la technique de programmation employée n'est pas assez fiable pour un usage dans des applications critiques, avouant implicitement que la fiabilité n'est pas un objectif. C'est un peu comme si un constructeur d'automobiles vendait ses voitures en prévenant que les véhicules pourraient ne pas

Des langages aux systèmes

La recherche française étudie beaucoup les problèmes fondamentaux que posent les systèmes répartis. Par exemple, l'utilisation croissante du *Web* pour des transactions commerciales (en particulier bancaires) imposent des protocoles de communication très sûrs : le projet *Verified Internet Protocol* (protocole certifié pour *Internet*) établit une «preuve formelle» de la confidentialité des protocoles de transactions proposés sur le réseau *Internet*. Citons aussi la conception de modèles théoriques des systèmes distribués, qui part du constat que la plupart des langages disponibles pour écrire des applications réparties intègrent mal les fonctions primitives donnant accès à la distribution : l'obtention des programmes mobiles et la résistance aux pannes locales nécessitent des appels système complexes. Inspiré de la théorie des «processus mobiles» de R. Milner, le J-calcul *(Join calculus)* apporte un modèle simple de la distribution qui définit précisément le comportement des primitives de distribution en présence de migrations et de pannes. À l'aide de ce calcul, on espère créer un langage de programmation de haut niveau intégrant directement la programmation distribuée et mobile.

Sur le plan moins spéculatif des systèmes d'exploitations, la recherche française a été à l'origine de *Chorus*, une famille de composants de systèmes d'exploitation qui permet à partir d'un ensemble réduit de fonctionnalités de base, de créer un système d'exploitation adapté aux contraintes de l'application répartie à réaliser. Les travaux actuels de la recherche en systèmes d'exploitation répartis concernent les mécanismes de partage d'informations dans les grands systèmes répartis, la tolérance aux pannes, en particulier dans la vidéoconférence, ainsi que l'application des techniques de haute disponibilité aux distributeurs de films vidéo à la demande : la défaillance d'un serveur pendant la diffusion du film doit rester imperceptible au spectateur qui conserve ainsi l'illusion d'un service continu.

Le système français concurrent du logiciel *Isis* cité dans l'article est le système *SafeTech* qui consiste en une couche logicielle intermédiaire qui intercepte les communications de l'application et assure la tolérance aux pannes sans modification du programme d'origine.

En ce qui concerne plus spécifiquement le système *Horus* que nous décrivent K. Birman et R. van Renesse, la recherche française a joué un rôle indirect dans sa conception : une version d'*Horus* a été écrite en *Caml*, un langage de programmation mis au point par les chercheurs de l'Institut national de recherche en informatique et en automatique. Ce langage est utilisé pour l'enseignement de l'option informatique dans les classes préparatoires aux Grandes Écoles. Les chercheurs américains ont pris le risque d'utiliser un langage de haut niveau pour écrire une application proche du système, et la qualité du logiciel obtenu montre que ce pari est gagné.

Si les informaticiens qui ont écrit *Horus/ML* (la version *Caml* du logiciel *Horus*) ont ainsi bénéficié du travail fait en France, en récupérant par le réseau *Internet* un langage de programmation adapté à leur problème, réciproquement, les développeurs et les usagers du langage *Caml* tirent bénéfice de l'expérience, puisque *Horus/ML* permet maintenant de créer et de gérer en *Caml* des systèmes répartis.

Cette synergie du travail d'équipes distantes de plusieurs milliers de kilomètres est caractéristique de la recherche actuelle. C'est bien entendu le réseau mondial d'information qui permet cette coopération. Dans le cas d'*Horus* et de *Caml* cette coopération porte directement sur l'amélioration de la fiabilité des réseaux qui permettront bientôt à tous de communiquer plus sûrement et plus facilement. Ainsi l'existence du réseau et des systèmes distribués rend possible ces recherches sur le réseau et les systèmes distribués, sujets et acteurs à part entière de la recherche qui les concerne...

Pierre WEIS – INRIA–Rocquencourt

fonctionner sur autoroute ! Les équivalents informatiques des ceintures de sécurité et des coussins gonflables sont rares, car les ingénieurs et les usagers s'intéressent plus à la rapidité et à la convivialité qu'à la fiabilité.

Lorsqu'on pense à la fiabilité, on imagine souvent des systèmes lents et pesants, à l'opposé de ces autoroutes de l'information tant vantées. Pourtant la technique qui rend les systèmes répartis robustes n'est pas nécessairement lente ni désagréable à utiliser. Alors que les informaticiens trouvent sans cesse de nouvelles utilisations aux liaisons entre les ordinateurs, la question s'impose : ces ponts supporteront-ils l'énorme flux d'informations qu'on se propose de leur faire transmettre ? Nous pensons que les systèmes répartis robustes offrent un outil précieux pour relier rapidement et de manière fiable des ordinateurs. Dans notre future société de l'information, ces réseaux offriront une nouvelle manière de concevoir les affaires et les loisirs. Toutefois, si un système réparti n'est pas construit pour fonctionner de façon robuste, il peut être préférable de ne pas le construire du tout.

Kenneth BIRMAN est professeur d'informatique à l'Université Cornell. Robbert van RENESSE est chercheur dans cette même université.

J. GRAY, J. BARTLETT et R. W. HORST, *Fault Tolerance in Tandem Computer Systems*, in *The Evolution of Fault-Tolerant Computing*, sous la direction de A. Avizienis, H. Kopetz et J.C. Laprie, Springer-Verlag, 1987.

Ivars PETERSON, *Fatal Defect : Chasing Killer Computer Bugs*, Random House, 1995.

R. BALTER, J.P. BANÂTRE, S. KRAKOWIAK, *Construction des systèmes d'exploitation répartis*, Collection Didactique, 1991, *INRIA-UCIS* diffusion.

Jean-Marie RIFFLET, *La communication sous Unix*, 7, Ediscience International,1990.

Informations disponibles sur le *Web* :

- Le programme *Horus* : http://www.cs.cornell.edu/Info/Projects/horus/
- Partage d'informations : http://prof.inria.fr/
- Le langage *Caml* :http://pauillac.inria.fr/caml/caml-fra.html
- Serveurs de films vidéo :http://www.inria.fr/Rapports/solidor/node21.html