
© HAMILTON RICHARDS, 2002

Un réseau d'ordinateurs peut-il prendre des décisions cohérentes ? Les travaux en algorithmique répartie ont abouti à des résultats qui permettent de dire quand cela est possible, et quand ça ne l'est pas.

Se mettre d'accord entre ordinateurs

Renaud Sirdey,
chercheur en informatique embarquée au sein du laboratoire d'intégration des systèmes et des technologies (List) du Commissariat à l'énergie atomique, à Saclay.
renaud.sirdey@cea.fr

Le réseau Internet, les processeurs « multicœurs », les grands systèmes de gestion de bases de données, les supercalculateurs : autant d'exemples de « systèmes répartis » à différentes échelles. Ce sont des entités composées d'un certain nombre d'unités de traitement, appelées sites, que l'on peut voir comme des ordinateurs indépendants, et qui sont interconnectés par des liens de communication – un réseau, en somme.

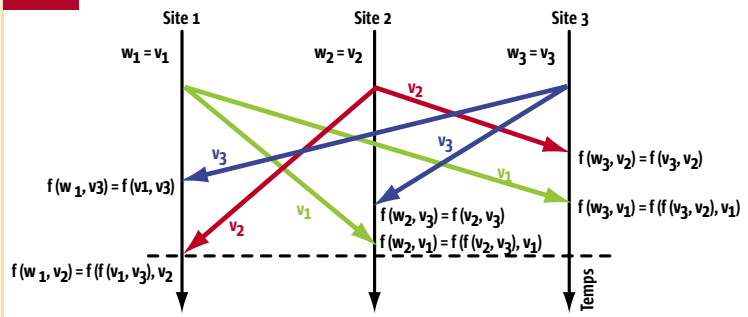
Sans même parler de performances, réussir à faire collaborer un ensemble d'ordinateurs pour qu'ils réalisent une tâche, tous ensemble et en toute fiabilité, n'est pas une chose aisée. La première raison est que les instructions des programmes qu'ils exécutent se trouvent entrelacées de manière indéterminée dans le temps. Dès lors, le nombre d'états potentiels du système est gigantesque, même lorsque les programmes en jeu sont très courts.

Ce problème fait l'objet d'une branche parmi les plus actives de l'informatique théorique : l'algorithmique répartie. On cherche à concevoir des algorithmes dont on peut prouver qu'ils fonctionnent correctement, et ce, dans un contexte où les sites et les liens de communication peuvent fonctionner à des vitesses différentes et imprévisibles (on parle de système réparti asynchrone) et où certains composants peuvent tomber en panne à tout instant.

À l'origine de l'algorithmique répartie, un pionnier de l'informatique : l'informaticien néerlandais Edsger Dijkstra (en médaillon). Physicien théoricien de formation, il a jeté les bases de la discipline au milieu des années 1960.

L'algorithmique répartie a permis de fournir deux grands types de résultats. Tout d'abord, pour certains problèmes, les chercheurs ont obtenu des algorithmes de résolution, ainsi que des mesures de leur complexité. Pour d'autres problèmes, ils ont établi au contraire des théorèmes d'impossibilité, c'est-à-dire des résultats indiquant que, pour un problème donné et dans un contexte donné, il n'existe pas d'algorithme permettant de le résoudre. Les deux sont le plus souvent directement utilisables par les ingénieurs. Et les seconds ne sont pas moins importants que les premiers : savoir qu'un théorème d'impossibilité s'applique à un système peut permettre de le repenser afin de contourner la difficulté ou, tout du moins, aider à appréhender les limites du système.

Fig.1 Le problème du consensus



DANS CETTE REPRÉSENTATION d'un système réparti à trois sites parfaitement fiables, les lignes verticales représentent l'écoulement du temps, et les lignes obliques symbolisent les transferts de messages. À la réception des messages, les sites effectuent des calculs pour mettre à jour leurs variables locales (les w_i). Chaque site envoie sa valeur initiale aux deux autres. En raison des propriétés particulières de f, l'accord intervient dès que tous les messages finissent par être reçus (ligne en pointillé).

LE SUPERCALCULATEUR BLUE GENE/L d'IBM compte plus de 130 000 processeurs. C'est un système réparti avec autant de sites qu'il faut faire travailler ensemble de manière fiable. © KIM KULISH/REA

Prenons l'exemple d'une transaction financière entre plusieurs banques, impliquant qu'un crédit soit accompagné d'un débit. Il s'agit d'un problème central en algorithmique répartie, celui dit du « consensus ». Soit tous les sites doivent appliquer la transaction, soit aucun d'entre eux ne doit le faire : il doit y avoir accord sur la validité de la transaction. Si ce n'est pas le cas (un débit sans crédit ou *vice versa*), alors la cohérence du système n'est pas garantie.

Sûreté et vivacité

Dans le problème du consensus, chacun des sites d'un système réparti est initialement doté d'une valeur, notée v_i. On cherche un algorithme qui vérifie les propriétés suivantes :

– tous les sites se mettent d'accord sur une valeur unique, notée v ;

– si tous les sites ont la même valeur initiale, $v_i = v$, alors la valeur d'accord est v ;

– l'accord entre les sites est obtenu en un temps fini.

Les deux premières propriétés sont des propriétés dites de sûreté, et la dernière est une propriété dite de vivacité.

Comment résoudre ce problème ? Chaque site doit d'abord envoyer sa valeur initiale v_i à tous les autres sites. À la réception des messages, les sites effectuent des calculs pour mettre à jour leurs variables locales, les w_i, en appliquant une fonction f comme f(x,y) = min(x,y). Si on prend l'exemple du premier site d'un système réparti à trois sites [fig. 1], w_1 prend successivement les valeurs v_1, puis $f(v_1,v_3)$, et enfin $f(f(v_1,v_3),v_2)$. La fonction f doit être commutative – f(x,y) = f(y,x) – et associative – f(x,f(y,z)) =f(f(x,y),z) – pour que la valeur de w_i à la réception du dernier message – donc au bout d'un temps fini – soit la même pour tous les sites. Pour le site 1, on a :

$w_1 = f(f(v_1,v_3),v_2) = f(v_1,f(v_3,v_2)) = f(v_1,f(v_2,v_3))$

Les propriétés de commutativité et d'associativité de la fonction f permettent de rendre le résultat final indépendant de l'ordre dans lequel les messages sont reçus, ordre que l'on ne maîtrise pas dans un système réparti asynchrone et qui est différent pour tous les sites.

On définit maintenant la tolérance aux pannes comme la capacité d'un système à fonctionner — même de manière dégradée — malgré la défaillance d'un ou de plusieurs de ses constituants. C'est un enjeu majeur en conception des systèmes répartis car dès lors que l'on considère un nombre suffisamment grand de composants, même s'ils sont très fiables, la probabilité pour qu'ils ne soient pas tous opérationnels à un instant donné devient vite non négligeable. Et cette nécessité de tolérance aux pannes est encore accrue dans le cas des systèmes toujours disponibles (les autocommutateurs d'un réseau téléphonique), ou lorsque la mission est longue (un grand calcul sur un supercalculateur).

C'est parfois pour accroître la tolérance aux pannes que l'on fait appel à des systèmes répartis. Il est, par exemple, courant qu'un système de gestion de base de données critique soit répété à l'identique sur un, voire plusieurs autres systèmes distants, afin de se prémunir contre les catastrophes naturelles.

Lent ou en panne ?

Mais envisager des pannes change singulièrement la donne. Revenons au problème du consensus. Sa résolution reste assez simple tant que tous les sites fonctionnent correctement. En présence de pannes, il n'existe tout simplement pas d'algorithme capable de résoudre le problème du consensus tel que nous l'avons énoncé. L'essence de la difficulté est que dans le modèle asynchrone de base, il n'est pas possible de faire la différence entre un site défaillant et un site temporairement très lent (par exemple en raison d'une surcharge temporaire). C'est là l'un des théorèmes fondamentaux de l'algorithmique répartie obtenu au milieu des années 1980 par les Américains Michael Fischer et Nancy Lynch, et le Britannique Michael Paterson.

Pour s'en sortir, on peut abandonner l'asynchronisme : tant que le nombre de défaillances est strictement inférieur au tiers du nombre de sites, le problème du consensus peut être résolu par un vote à la majorité portant, pour chaque site, sur l'ensemble des valeurs reçues (ou en prenant la valeur médiane dans le cas de variables non binaires) dans le cas d'un système réparti synchrone – c'est-à-dire un système réparti où tous les sites ont accès à une horloge commune et où le temps maximal de transfert d'un message est connu *a priori*. Le problème est qu'à l'heure des infrastructures réseaux intrinsèquement asynchrones, tels les réseaux IP, il est difficile de construire des systèmes répartis synchrones à grande échelle. À moins d'être prêt, entre autres, à équiper chaque site d'un récepteur GPS ou d'une horloge à rubidium. ■■

POUR EN SAVOIR PLUS

■ Michel Raynal, *La Communication et le temps dans les réseaux et les systèmes répartis*, Eyrolles, 1991.