Chapitre 1

Introduction aux systèmes répartis

Ce chapitre définit le domaine des systèmes répartis. Il permet d'apprendre à reconnaître un système réparti d'un système parallèle ou concurrent, ainsi qu'à relever les principales caractéristiques du système. Il liste enfin les problématiques du domaine.

1.1 Définition

La notion de système réparti émerge dès les balbutiements de l'informatique, et transparaît dans les travaux de Von Neuman, qui imaginait des calculs réalisés par plusieurs entités calculant et communiquant. Mais l'étude des systèmes distribués débute dans les années soixante. À cette époque, le développement de l'usage de l'ordinateur accroît le besoin de communication entre les machines. En outre, pour augmenter la puissance des ordinateurs, l'idée de les connecter se développe. Le besoin de communication d'une part, et celui de calcul d'autre part, conduit donc à faire coopérer des machines distantes les unes des autres par le biais de réseaux de télécommunication.

Un système distribué peut être défini comme un réseau d'«entités calculantes» ayant le même but commun : celui de la réalisation d'une tâche globale à laquelle chaque entité contribue par ses calculs locaux et les communications qu'elle entreprend, sans même qu'elle ait connaissance du dessein global du système.

Certains auteurs font la distinction entre les termes *systèmes distribués* et *systèmes répartis*. Le premier, de l'anglais *distributed systems*, induirait l'idée d'une distribution des tâches effectuée par un coordinateur (*e.g.*, site central), tandis que le second supposerait une coopération des tâches, en vue de la répartition du travail. Il ne nous paraît pas nécessaire d'établir une telle distinction dans ce cours. La littérature anglaise ne fait d'ailleurs pas cette différence, et parle de *distributed systems*, où le terme *distributed* signifie «réparti dans l'espace».

La notion de système «distribué», c'est-à-dire réparti dans l'espace, est relative. On imagine aisément que les programmes faisant intervenir des ordinateurs distants à l'échelle de l'homme reposent sur des algorithmes répartis (*e.g.*, routage des messages dans Internet). Mais pour les concepteurs d'un ordinateur, la carte mère est également composée d'éléments distribués et coopérant par des communications (au sens large) : processeur(s), caches, mémoire, contrôleurs de disques *etc.* Pour l'utilisateur final de l'ordinateur, la machine apparaîtra comme un système non distribué. En effet, le but du domaine des systèmes distribués est de donner à l'utilisateur une vue non distribuée d'un environnement composé d'éléments répartis dans l'espace, et communicant les uns avec les autres.

Il faut distinguer ici l'appréciation par l'utilisateur de la distance (qu'indiquerait une certaine lenteur de l'application par exemple) de celle de la répartition. Lorsque l'accès à la liste des fichiers d'un

répertoire est longue, l'utilisateur en déduira qu'ils ne sont pas stockés sur le disque dur de sa machine. Mais il ne faudrait pas que le comportement de l'application l'oblige à prendre en considération le fait que les fichiers sont gérés par plusieurs serveurs distants coopérant entre eux. C'est à l'application répartie – ici de gestion de fichiers distants – de masquer cette répartition et la complexité qu'elle engendre.

1.2 Différents systèmes distribués

Les systèmes distribués peuvent être distingués selon les hypothèses qu'ils admettent : nombre de sites connu ou non, topologie connue ou non, codes uniformes (identiques d'un nœud à un autre) ou non, évolutions concertées (synchronisations) des sites ou non, *etc.* Au niveau des communications, on se demandera si le temps de délivrance d'un message est incertain ou bien fixé, si l'ordre des messages sur un lien est connu ou aléatoire, *etc.* On distingue généralement quatre paramètres essentiels : la topologie du réseau, la synchronisation des nœuds, les pannes tolérées et le type de communications.

Un site est essentiellement caractérisé par son programme, ses variables (et la place mémoire qu'elles requièrent, *i.e.*, nombre d'états ou quantité de bits), et par sa connaissance de l'extérieur (possibilité de distinguer ses voisins ou non, par exemple). Selon les auteurs — et les contextes —, un site est appelé *processeur*, *entité calculante*, *processus*, *nœud*, *agent*, *etc.* Pour un site, la seule façon de recevoir de l'information sur l'état du système ou sur une autre entité consiste à recevoir un «message» au sens large, c'est-à-dire de l'information sous une forme qui dépend de l'implémentation ou du modèle considéré.

Le réseau est constitué des liens qui véhiculent les messages d'une entité calculante à une autre. Il s'agit là aussi d'un terme générique désignant les moyens de communication, qui sont bien différents d'une implémentation ou d'un modèle à un autre. Pour caractériser le réseau, on utilise des paramètres qualitatifs, tels que la topologie, la connectivité, la synchronisation des envois de messages, les communications uni- ou bi-directionnelles, *etc.* Les paramètres quantitatifs précisent le diamètre du réseau, sa bande passante, sa latence, la taille maximale d'un message, *etc.*

L'accès éventuel à une certaine connaissance globale du système distribué est un paramètre important. Il peut s'agir d'une horloge globale, d'une numérotation des entités, d'un état global périodiquement distribué (synchronisation, terminaison, *etc.*), de l'identifiant[(a)] d'un nœud particulier (élection d'un *leader*), *etc.* On distingue ainsi les systèmes dont chaque nœud connaît tous les autres, des systèmes où les nœuds n'ont, au contraire, qu'une connaissance partielle des autres. L'utilisation d'un *routage* permet d'émuler le premier cas par le second. L'identifiant des nœuds est un paramètre central. Lorsqu'il n'y a pas moyen de distinguer deux nœuds quelconques grâce à un identifiant ou à un comportement (exécution) différent, on parle de système *anonyme*. Un certain nombre de problèmes classiques deviennent insolubles dans un système anonyme. Par exemple, il a été montré que, sauf à avoir recours à des méthodes probabilistes, il est impossible d'élire un site faisant office de chef (*leader*) dans les systèmes anonymes (*cf.* chapitre 9).

Le mode de communication est également une caractéristique importante d'un système distribué. On distingue les communications par échange de messages des communications par mémoire partagée, ou registres. Il faut aussi préciser si les tampons de réception sont de taille finie ou non, et si les messages peuvent être désordonnés ou non (propriété FIFO[1]). En effet, dans un réseau de télé-

1. *First In First Out.*

[(a)] L'identifiant est aussi appelé *identité* ; il s'agit du nom d'un site.

communication, si les communications ne sont pas effectuées en mode connecté, alors les messages peuvent arriver par des routes différentes, et dans un ordre différent de celui d'émission.

Nous revenons plus précisément sur les paramètres caractérisant un système distribué au chapitre 2 page 11.

1.3 Particularités des systèmes distribués

1.3.1 Calcul parallèle, concurrent et distribué

Programmation parallèle. Une *machine parallèle* est un ordinateur possédant plusieurs unités de calcul coopérant dans le but d'exécuter une tâche commune, le programme parallèle. Les caractéristiques principales de ces machines concernent l'autonomie de leurs processeurs (*i.e.*, leur indépendance), le type de mémoire, et le réseau d'interconnexion des processeurs.

Ainsi, les machines dites *MIMD*[2] ont des processeurs autonomes, ayant chacun leur mémoire. Même si des recherches sont menées sur des techniques de programmation plus pratiques, la majeure partie de ces machines sont actuellement programmées par passage de messages, à l'aide de bibliothèques de communication telles que PVM[3] ou MPI[4]. Il en est de même des réseaux de stations, ou bien encore des grappes de PC (et plus généralement du *calcul sur grille*[5]).

On retrouve alors des processus indépendants, ne pouvant communiquer que par envoi de messages, et coopérant pour l'achèvement d'une tâche globale. Bien que la problématique soit voisine de celle des systèmes distribués, on parlera d'*algorithmique parallèle* et non d'algorithmique répartie.

2. *Multiple Instruction, Multiple Data*, terme de la classification de Flynn, 1956.
3. *Parallel Virtual Machine*
4. *Message Passing Interface*
5. En anglais, *grid computing*

Programmation concurrente. De même, dans un système d'exploitation multi-tâches, les processus évoluent de façon autonome, et communiquent entre eux. Le programmeur est alors confronté aux problèmes des synchronisations de processus, des inter-blocages, de l'exclusion mutuelle, *etc.* Ces problèmes relèvent également des systèmes distribués. Néanmoins, on parlera de *programmation concurrente* et non de programmation distribuée.

Motivations et conséquences. La différence essentielle entre la programmation distribuée et la programmation parallèle ou concurrente réside dans la raison de la distribution des données, et dans ses conséquences.

Dans le cas de la programmation sur machine parallèle, les données sont volontairement réparties sur un ensemble de processeurs, afin d'obtenir la puissance de calcul désirée. Mais pour ne pas nuire aux performances, on utilise alors un réseau dédié particulier, permettant des échanges de données efficaces (cas d'une machine parallèle). Sans cet équipement, la distribution des données aurait été jugée inintéressante. Certaines machines possèdent en outre des mécanismes matériels permettant de centraliser rapidement une information ou de synchroniser les différents processeurs. Le réseau n'est donc pas subit ; il est voulu.

Dans le cas d'une programmation sur réseau de stations, la motivation est souvent celle d'une parallélisation à faible coût, sans équipement spécifique. Le réseau est plus contraignant. Cependant des dispositifs logiciels permettent de faciliter la programmation, tels qu'un processus maître aidant à la concentration d'une information (fin du programme, synchronisation, *etc.*), ou bien encore un système de fichier commun, une horloge commune, *etc.* Fournir à une application programmée sur un réseau de stations ce type de facilités relève de la programmation répartie en ce sens qu'elles nécessitent un algorithme réparti. Mais une application bénéficiant de ces facilités est généralement considérée comme une application parallèle et non distribuée.

Notons que dans ces systèmes parallèles, les pannes sont rarement gérées. Lorsqu'elles le sont, il s'agit en fait d'inclure au programme parallèle une reprise sur erreur grâce notamment à des sauvegardes régulières. On part du principe qu'une panne est facilement détectée, et qu'il vaut mieux relancer le programme parallèle plutôt que de continuer dans une configuration dégradée[(b)].

Enfin, dans le cas d'un système d'exploitation, l'unique noyau et la mémoire centrale (et le verrouillage du bus mémoire) facilitent grandement la coordination des différents processus. En particulier, la mémoire partagée permet de gérer des *sémaphores*, sorte de compteurs qui peuvent être consultés et modifiés au cours d'une seule opération (*cf.* chapitre 14 page 141), et qui permettent de résoudre la plupart des problèmes de la programmation concurrente. Là encore, la contrainte de distribution des données et des traitements sur les processus est faite dans un but d'efficacité, et à la condition qu'un système particulier aidant à la centralisation des informations, permette de garantir cette efficacité recherchée.

Cas des systèmes distribués. Comparativement à la programmation concurrente ou parallèle, la caractéristique principale d'un système distribué réside dans le caractère intrinsèquement distribué des données manipulées. La distribution n'est pas désirée ; elle est subie. Certes, certains programmes de calcul sont parallélisés parce qu'ils nécessitent plus de mémoire que ne peut en offrir une seule machine. Et à l'inverse, certaines applications sont réparties dans le seul but d'augmenter leur robustesse. Mais dans le cas général, la répartition des utilisateurs et des données dans une application répartie ne relève pas d'un choix du programmeur contrairement à la programmation parallèle : c'est un état de fait qu'il doit prendre en compte.

En conséquence, les vues d'ensemble du système sont difficiles à construire. Il n'existe pas d'emblée de dispositif (matériel ou logiciel) permettant de centraliser les informations. Les pannes sont délicates à détecter, et il est souvent préférable de poursuivre l'application distribuée malgré des défaillances réelles ou supposées plutôt que de réinitialiser l'ensemble du système. Ces réinitialisations sont souvent longues et problématiques pour les utilisateurs. Ainsi, s'il est possible de relancer un calcul parallèle à la suite d'une panne, on envisage guère de relancer tous les routeurs lorsque l'un d'eux est en panne.

Autre conséquence de cette distribution subie, les valeurs de certains paramètres sont très différentes entre un système parallèle ou concurrent et un système distribué. En particulier, dans un système distribué, le temps de calcul est souvent supposé nul comparativement au temps de communication. Le contraire est souvent admis dans un système concurrent, tandis que le rapport *temps de calcul* sur *temps de communication* est une caractéristique importante d'un programme parallèle, orientant la façon de le paralléliser (*cf.* figure 1.1 page ci-contre). Pour optimiser les programmes parallèles, on tente de recouvrir les communications par les calculs, c'est-à-dire que le programme parallèle est organisé de telle manière que les processeurs calculent en attendant les communications. Cette optimisation n'est pas évoquée en algorithmique répartie.

Conclusion. Ainsi, même si les domaines de la programmation parallèle, concurrente et distribuée partagent certains concepts communs, on peut néanmoins les distinguer, de par la motivation de la distribution des données, et ses conséquences en terme de facilité de programmation ou de valeurs

[(b)] Des techniques de résistance aux pannes ont cependant été imaginées dans certains multi-processeurs comportant un grand nombre de processeurs dans un seul circuit intégré, ou bien dans une structure suffisamment coûteuse ou complexe pour qu'il ne soit pas envisageable de changer l'élément matériel défaillant. Certains longs calculs parallèles peuvent ainsi se poursuivre malgré la panne de quelques éléments de la machine, grâce, par exemple, à un routage évitant les éléments défaillants du réseau.

FIG. 1.1 – Ratio *temps de calcul* sur *temps de communication* et type de programmation.

des paramètres principaux. Plus le système présentera des facilités (synchronisation, horloge commune...), moins sa programmation sera qualifiée de répartie.

1.3.2 Systèmes séquentiels, centralisés et distribués

Système séquentiel. Un système est *séquentiel* lorsqu'il n'est composé que d'une seule entité. On peut connaître aisément l'état dans lequel se trouve un tel système à un instant donné. En outre, les différentes actions réalisées peuvent être classées (ordonnées) par leur date d'exécution. Il existe une relation d'ordre totale entre deux actions, l'une étant antérieure à l'autre. L'exécution d'un programme est *déterministe*, c'est-à-dire qu'il donne toujours le même résultat si on lui fournit les mêmes entrées[(c)].

Système centralisé. Un système est *centralisé* lorsqu'il est composé de plusieurs entités communiquantes, ne pouvant fonctionner qu'en faisant référence à un site particulier, investi d'un rôle plus important que les autres. Ce dernier agit comme un chef d'orchestre, et assure le fonctionnement de l'ensemble du système. L'existence d'un site central dans un système réparti facilite son utilisation car certains dispositifs permettront de connaître l'état global du système, ou d'ordonner les actions. Il est donc plus facile de construire un système déterministe. Par exemple, si seules les requêtes indépendantes sont exécutées simultanément sur le site central, alors l'état du système peut être déterminé. L'ordre sur les actions n'est cependant pas total car on ne pourra pas comparer deux requêtes indépendantes. Si les requêtes sont exécutées une par une (sérialisation), alors l'ordre induit par les dates d'exécution des tâches sera total.

Système réparti. Cependant, l'utilisation d'un site privilégié n'est pas nécessairement une bonne idée dans un système géographiquement réparti, car son éventuelle défaillance bloquerait l'ensemble du système, qui peut être vaste et servir à de nombreux utilisateurs. En outre, la remise en route de l'ensemble du système serait subordonnée à une intervention sur le site privilégié, ce qui peut être un problème dans un système géographiquement étendu. Signalons également le phénomène de *goulot d'étranglement*[6], qui survient dans un système lorsque celui-ci est trop dépendant d'une ressource particulière, d'un axe de communication, d'un serveur *etc.* Dans ce cas, l'application peut être fortement ralentie, à la manière de la circulation dans une ville dont l'un des axes principaux serait surchargé. On rencontre ce problème dans certaines applications parallèles, dont le contrôle est trop dépendant d'un site privilégié vite débordé. L'une des solutions possibles consiste alors à distribuer le contrôle de l'application sur plusieurs sites.

6. En anglais, *hot spot.*

Conclusion. Ainsi, l'absence de site central est une garantie de sécurité, et dans certains cas de performance. Mais elle complique la conception d'algorithmes répartis déterministes.

[(c)] Dans le cas où le programme utilise une variable aléatoire, il faut considérer la valeur de cette variable comme l'une des entrées du programme.

1.4 Problématiques des systèmes distribués

Problèmes à surmonter. La conception d'algorithmes répartis se heurte à plusieurs difficultés. La coordination de sites distants les uns des autres, l'exploration d'un réseau, ou la centralisation d'une information ne sont pas immédiates. L'état du système peut évoluer au cours de ces tâches que l'on aurait voulu « atomiques ». Dans certains cas, les communications elles-mêmes – pourtant indispensables à tout algorithme – constituent des algorithmes répartis. Par exemple, comment partager un bus de communication à accès unique entre plusieurs sites, sachant qu'ils ne peuvent communiquer pour se coordonner ? Bien souvent, les applications réparties nécessitent que les sites prennent une décision commune. Comment par exemple élire un site coordonnateur ou bien converger vers une valeur unique sur chaque site lorsque chacun d'eux pourrait prendre une décision différente et que le système est non déterministe ? De même, l'utilisation de ressources partagées entre les sites nécessite des algorithmes répartis d'allocation, d'exclusion mutuelle, de gestion des interblocages... Similairement, l'utilisation de données partagées requiert des algorithmes de gestion des réplicats. Cette gestion serait simplifiée si les sites avaient accès à une horloge commune. Cependant la distribution du temps dans le réseau repose sur des algorithmes répartis, de même que la synchronisation entre les sites.

Recherche. Les systèmes répartis ont donné lieu à de nombreux travaux de recherche depuis les années soixante, mais le domaine s'est surtout développé à partir des années quatre-vingt. Les travaux publiés ne reposent pas tous sur les mêmes hypothèses, et la question du modèle – chargé de représenter la réalité tout en offrant une abstraction intéressante – est un problème en soit. L'implémentation implique souvent des compromis, car certaines hypothèses ne sont pas vérifiées. C'est par exemple le cas en ce qui concerne la tolérance aux défaillances (perte de messages, panne d'un site...), d'autant que des résultats d'impossibilité ont été montré. Il est difficile de résumer les acquis du domaine. Après s'être intéressé aux réseaux fixes, les recherches se sont orientées vers les réseaux mobiles. La dynamique de ces réseaux pose de nouvelles difficultés de part l'instabilité de la topologie. Les techniques permettant de masquer la répartition des sites, et la concurrence de leurs exécutions sont maintenant acquises. Par contre, masquer la mobilité ou la dynamique d'un réseau reste encore du domaine de la recherche. Par ailleurs, la tolérance aux défaillances est l'objet d'intenses recherches, d'autant que les réseaux de mobiles ou de capteurs nécessitent des algorithmes robustes pour continuer de fonctionner malgré la panne de certains éléments (*e.g.*, pannes de batteries).

Plan du cours. La modélisation des systèmes réparties est développée dans la partie I page 11. Le modèle de communication à passage de messages est privilégié dans ce cours car il est plus proche de la réalité. Ce parti pris impacte inévitablement le choix des algorithmes présentés. Les concepts fondamentaux sont également développés dans cette première partie : causalité, coupure, horloges logiques et étas globaux.

Les difficultés liées à la répartition des informations et des centres de décisions sont abordés dans la partie II page 71. On y étudie les communications, l'exploration des réseaux, l'élection d'un coordonnateur et la détection de la fin d'une application répartie.

Les problématiques liées aux évolutions concurrentes des sites sont développées dans la partie III page 117. L'allocation des ressources, les interblocages et l'exclusion mutuelle répartie y sont présentées. Puis la gestion des données réparties est détaillée.

Enfin, la gestion de la robustesse est présentée dans la partie IV page 179. Il s'agit des algorithmes et techniques permettant de rendre sûres les applications réparties face aux pannes involontaires et aux agressions volontaires. La distribution du contrôle à l'ensemble des sites d'une application la rend plus vulnérable aux difficultés d'un milieu hostile.

1.5 Conclusion

Résumé. Les systèmes répartis sont nés de la connexion des ordinateurs entre eux. Mais plus généralement, un système réparti est constitué d'entités calculantes, coopérant en s'échangeant de l'information afin de réaliser un but commun. La notion de répartition dans l'espace est relative ; on trouve des algorithmes répartis depuis les circuits électroniques jusqu'aux applications fonctionnant au dessus d'un réseau de télécommunication tel qu'Internet. Le but du domaine des systèmes répartis est de masquer la répartition des données et des centres de décision aux utilisateurs.

Les divers systèmes distribués se distinguent par des paramètres qualitatifs et quantitatifs, ainsi que par les facilités qu'ils offrent ou, au contraire, qui leur font défaut. Parmi celles-ci, la connaissance globale (nom des sites, routage, horloge, synchronisation, *etc*.) dont un site peut disposer est un paramètre important.

Les systèmes distribués sont plus complexes à appréhender que les systèmes centralisés et séquentiels car l'ordre des actions est plus difficilement déterminé. Les systèmes distribués se distinguent des systèmes parallèles et concurrents par la distribution intrinsèque des données et des unités de transformation de ces données. La répartition des données et des unités de calcul est subie, et le but est de masquer cette distribution, en offrant aux utilisateurs les outils nécessaires pour l'oublier, et ainsi faciliter l'utilisation du système.

Pour concevoir une application répartie, il est nécessaire de surmonter les difficultés liées à la répartition des centres de décision autonomes (sites) et des données, à l'évolution concurrente des sites, aux pannes et agressions rendues plus criantes dans le contexte réparti. Si l'on sait maintenant masquer la répartition dans les réseaux peu mobiles, la conception d'applications réparties adaptées aux réseaux dynamiques reste encore du domaine de la recherche.

Lectures complémentaires. On trouvera dans [2] une introduction aux systèmes répartis orientée vers les ordinateurs communiquants. Des précisions sur le terme *distributed* (« *spread out accross the space* ») sont données dans [3] ; les auteurs discutent également des principaux paramètres permettant de classer les systèmes répartis. Dans [4], l'auteur classe les systèmes selon la synchronisation des nœuds, et le type de communication.

Des exemples d'architectures parallèles sont présentées dans [5, 6] ; une taxonomie est présentée dans [7].