

# Précis de répartition

## Définition et problématique

3ième Année Informatique et Mathématiques Appliquées

11 octobre 2004

### Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Les épines...</b>	<b>3</b>
2.1	Pas d'état global	3
2.2	Pas d'horloge globale	3
2.3	Fiabilité relative	4
2.4	Sécurité relative	4
2.5	Non-déterminisme	4
<b>3</b>	<b>Les parfums...</b>	<b>4</b>
3.1	Partage et Mise à disposition	4
3.2	Répartition géographique	5
3.3	Autres avantages	5
<b>4</b>	<b>La recherche de solutions (Conceptualiser, comprendre, expérimenter)</b>	<b>6</b>
4.1	La théorie	6
4.2	L'algorithmique	6
4.3	Les langages de programmation	6
4.4	Les systèmes d'exploitation	7
4.5	Les environnements d'exécution répartie (middleware)	7
<b>5</b>	<b>Principes de conception</b>	<b>8</b>
5.1	Transparence d'accès	8
5.2	Transparence de localisation	9
5.3	Transparence du partage	9
5.4	Transparence de la réplication	10
5.5	Transparence des fautes	10
5.6	Transparence de la migration	10
5.7	Transparence de charge	10
5.8	Transparence d'échelle	10
<b>6</b>	<b>Exemple de répartition : Variations sur un thème de location</b>	<b>11</b>
6.1	Exposition du problème	11
6.2	Le contrôle des mouvements de véhicules	11
6.2.1	Répartition par accès distant (maintien d'un chef d'orchestre)	12
6.2.2	Répartition par image (deux solistes mais ils s'observent)	12
6.2.3	Répartition par circulation (une partition nomade)	13

6.3	Enrichissement du système (Plusieurs orchestres) . . . . .	14
6.3.1	Répartition par observation et circulation . . . . .	14
6.3.2	Répartition par réplication . . . . .	15
6.4	Conclusion . . . . .	15
6.5	Exercices . . . . .	15

# 1 Introduction

La répartition des applications informatiques fait l'objet d'un fort développement depuis une vingtaine d'années. Cet essor s'est accentué avec les progrès technologiques des réseaux de télécommunication. En effet, les applications réparties doivent leur naissance en tout premier lieu au mariage de l'informatique et des télécommunications.

Un point anecdotique concerne la terminologie. Doit-on parler de systèmes répartis ou de systèmes distribués. Les constructeurs d'ordinateurs ou de réseaux utilisent plutôt le terme "distribué". Les spécialistes du logiciel préfèrent le terme "réparti". Appartenant à cette confrérie, j'utiliserai donc le mot répartition et ses dérivés. De toute façon, l'objet de ce petit précis est justement de donner une sémantique précise à ces termes.

## 2 Les épines...

La problématique de la répartition est née avec l'idée de faire communiquer des ordinateurs via un réseau de communication. Par simple échange de messages, il est en effet possible de développer des applications qui peuvent disposer de l'ensemble des ressources de l'architecture composée des nœuds "ordinateurs" et du réseau.<sup>1</sup> La puissance et l'intérêt d'une telle architecture répartie est évidemment étroitement liée à la capacité de communication offerte par le réseau. De ce point de vue, les progrès technologiques réalisés en la matière ont largement contribué au développement rapides des applications réparties.

Reste le problème du développement, de la programmation des applications réparties. En effet, la conception d'applications réparties a soulevé de nombreux problèmes. Aujourd'hui, des principes et concepts, mais aussi des langages, des systèmes d'exploitation, des environnements sont disponibles. Ils sont le résultat d'une vingtaine d'années de maturation à travers des centaines de projets de recherche. Tous les problèmes ne sont pourtant pas encore résolus et les applications futures soulèveront sans doute de nouveaux défis.

Une question essentielle est de percevoir pourquoi un tel effort de recherche a été nécessaire. Autrement dit, pourquoi et en quoi la programmation d'applications réparties se distingue d'une application "centralisée" ?

L'origine des difficultés tient à deux hypothèses perdues et à deux propriétés affaiblies.

**Pas d'état global**  
**Pas d'horloge globale**  
**Fiabilité relative**  
**Sécurité relative**

### 2.1 Pas d'état global

Dans une architecture répartie, un nœud ne possède pas une connaissance immédiate exacte de l'état d'un autre nœud. En effet, cette connaissance passe par l'échange d'un message qui introduit obligatoirement un délai. Un nœud connaît son état local et n'a qu'une connaissance du passé (certes parfois très proche) des autres. Cette difficulté conduira à la recherche d'algorithmes qui permettent de construire des clichés globaux qui représenteront un état passé possible de l'application.

### 2.2 Pas d'horloge globale

Chaque nœud possède sa propre horloge pour dater les événements qui lui sont locaux. Par conséquent, si les horloges indépendantes de chaque nœud ne sont pas parfaitement synchronisées, l'ordre des événements répartis dans l'application n'est pas déductible à partir des datations locales. Cette difficulté conduira à définir des datations logiques qui permettent de corriger ce problème.

---

<sup>1</sup>Il faut bien remarqué que cette définition exclut une architecture composée d'un ordinateur et de terminaux de saisie/affichage, même si ceux-ci sont distants.

Ces deux propriétés augmentent fondamentalement la difficulté de compréhension et d'analyse des applications réparties.

Deux autres propriétés doivent être prises en compte, en l'occurrence, la fiabilité et la sécurité.

### 2.3 Fiabilité relative

Une architecture répartie est composée de nœuds actifs dont le fonctionnement correct est relativement indépendant. Si deux nœuds sont simplement reliés par un réseau de communication, l'arrêt de l'un est généralement indépendant de l'arrêt de l'autre.<sup>2</sup> Contrairement aux systèmes centralisés, un système réparti peut donc tolérer la défaillance de certains de ses éléments et continuer à assurer certains services à ses usagers. Cependant, l'arrêt d'un site parmi un nombre élevé, contrairement à une unité isolée, n'est pas un événement exceptionnel. Par conséquent, il faudra que le concepteur prenne en compte ce risque non-négligeable de défaillance. Une architecture répartie est une bonne base pour développer une application tolérante aux fautes, mais le traitement des défaillances doit y être soigneusement étudié.

### 2.4 Sécurité relative

Une architecture répartie est plus difficile à protéger contre les malversations qu'une architecture centralisée. Ceci pour deux raisons :

- les points d'accès aux ressources du système global sont multiples et souvent "hors les murs". Autrement dit, l'utilisateur doit être authentifié.
- le réseau de communication est un point d'accès potentiel pour toute tentative d'intrusion.

Plus généralement, les nœuds connectés peuvent apparaître et disparaître dynamiquement et la configuration globale évoluer dynamiquement au cours du temps. Ceci peut conduire à ne pas connaître exactement la totalité de l'architecture répartie. Cette tendance est de plus en plus forte avec les possibilités de connexion via des ordinateurs portables.

### 2.5 Non-déterminisme

Enfin, il ne faut pas oublier qu'une application répartie est aussi une application parallèle asynchrone. Les nœuds s'exécutent en parallèle et chaque nœud peut lui-même comporter plusieurs activités parallèles (processus, threads). Il existe ainsi un non-déterminisme, une non-reproductibilité et une explosion combinatoire des états qui constituent les difficultés classiques de la programmation parallèle asynchrone.

## 3 Les parfums...

Après ce qui précède, peut-on encore trouver un intérêt à la répartition ? Malgré les handicaps précédents, la réponse est oui. Il suffit de lister les principaux avantages.

### 3.1 Partage et Mise à disposition

La répartition est avant tout un moyen de mise à disposition et donc de partage de ressources et services. Cette idée de partage et de disponibilité constitue même la sémantique que l'on peut donner au mot répartition.

#### Exemples

Une ressource telle qu'une imprimante peut être rendue accessible à tous les usagers d'une architecture répartie. Elle peut même constituer un nœud du réseau.

---

<sup>2</sup>Excepté quelques cas très particuliers : coupure électrique globale concernant des ordinateurs en réseau local.

Des fichiers localisés sur un disque d'un nœud peuvent être rendus visibles voire modifiables par des usagers connectés à un nœud distant. Un service de gestion de fichiers "répartis" est un des services de base des systèmes d'exploitation répartis.

### 3.2 Répartition géographique

La répartition est un moyen essentiel pour mettre à disposition des usagers les moyens informatiques locaux dont ils ont besoin tout en gardant la possibilité d'accéder aux ressources et services distants de leurs collègues.

#### Exemples

Un système de réservation (d'hôtels, d'avions, de trains, ou de tout à la fois) est un exemple de système qui peut avoir plusieurs centres dans différents lieux, par pays, par compagnie, par agences.

Un système bancaire peut comporter un ensemble d'agences régionales qui communiquent avec le siège central.

Le contrôle aérien peut nécessiter un découpage en secteurs dont il faudra coordonner les interactions.

### 3.3 Autres avantages

Moins essentiels mais non négligeables, on n'oubliera pas les autres points suivants :

- **Puissance de calcul** La connexion de machines en réseau permet d'obtenir à moindre coût une puissance de calcul importante. Bien entendu, l'exploitation optimale de ces performances potentielles nécessite de paralléliser les algorithmes de calcul et de disposer d'un environnement d'exécution répartie adéquat. En la matière, quelques produits de ce type sont aujourd'hui largement diffusés (PVM, MPI par exemple).
- **Disponibilité** La disponibilité d'un service développé sur une architecture répartie peut être rendue plus grande que celle d'un service centralisé. La relative indépendance des défaillances qui peuvent survenir dans les nœuds, permet de continuer à offrir un service même dégradé. En particulier, la réplification d'un même service par l'installation de plusieurs serveurs équivalents est une solution classique mais de mise en œuvre délicate notamment si les serveurs sont à état rémanents.
- **Flexibilité** Une architecture répartie est par nature modulaire. Il est donc plus facile d'ajouter ou d'enlever un nœud connecté au réseau, le système global continuant à être disponible. Cette caractéristique tend à jouer un rôle important avec la mobilité de plus en plus grande des usagers. En effet, ce sont maintenant des ordinateurs portables qui se connectent par un réseau commuté pour accéder à un système réparti aux frontières parfois mal connues (par exemple le réseau Internet). Les points d'accès sont donc mobiles. On parle d'informatique nomade. La connexion est temporaire et doit être même minimisée à cause des coûts de communication non négligeables. Les problèmes posés concernent donc essentiellement la localisation, l'identification et l'authentification des usagers "mobiles" et l'optimisation des temps de connexion (par l'utilisation d'agents mobiles par exemple).

Cette flexibilité se situe aussi au niveau logique. Un nouveau service peut être installé sur un nœud sans nécessiter une reconfiguration des autres nœuds.

Autrement dit, un système réparti peut en principe mieux s'adapter d'une part, à une croissance des besoins informatiques d'une entreprise en permettant une adaptation dynamique et continue et d'autre part, au nombre variable et à la mobilité de ses usagers.

Les avantages précédents ont fait le succès des systèmes répartis. Ils permettent aussi de donner une définition plus précise du mot répartition.

La répartition est la mise à disposition d'un ensemble de ressources et services connectés via un réseau pour tous les usagers qui possèdent un droit d'accès en un point quelconque.
---

## 4 La recherche de solutions (Conceptualiser, comprendre, expérimenter)

La recherche de solutions face aux difficultés intrinsèques posées par la répartition s'est déployée dans différents domaines :

### 4.1 La théorie

Pour bien comprendre la problématique de la répartition, il est important de pouvoir modéliser le plus formellement possible sa sémantique. Il s'agit donc d'élaborer des modèles de calcul traduisant de façon abstraite et formelle (mathématique) les propriétés d'un calcul réparti.

Une première école a pour origine les travaux de Milner et Hoare. Elle s'appuie sur la définition d'algèbres de processus communicants. Un calcul réparti est un terme algébrique qui décrit un ensemble de processus qui sont autant de termes composants. La communication entre processus apparaît sous la forme d'un rendez-vous, c'est-à-dire d'une communication synchrone (tampon de taille nulle) point à point la plus élémentaire.

Une deuxième école s'appuie sur le concept d'acteur. Le protocole de communication entre acteurs est asynchrone, chaque acteur possédant une boîte à lettres pour recevoir les messages des autres acteurs. Le typage des messages permet d'associer un comportement "réactif" spécifique et modifiable dynamiquement à la réception de chaque message. Ce modèle d'acteurs communiquant par message a lui aussi été formalisé.

Enfin, les systèmes de transitions et la logique temporelle sont aussi utilisables et utilisés pour spécifier et modéliser des traitements répartis. Dans ce cas, la communication est souvent modélisée simplement sous la forme de séquences de messages.

### 4.2 L'algorithmique

L'écriture d'algorithmes adaptés à une architecture répartie soulève des problèmes algorithmiques spécifiques. Ceux-ci ont pour origine les deux hypothèses "perdus" bien utiles au programmeur. Un courant de recherche s'est donc développé pour étudier et proposer des algorithmes répartis apportant en particulier des solutions à des problèmes génériques.

Une importante classe d'algorithmes concerne la définition de protocoles de communication point à point ou à diffusion. En effet, pour pouvoir coopérer via des échanges de messages, les partenaires doivent se mettre d'accord sur le séquençement et le type de ces messages. La définition de nouveaux protocoles de communication est donc une activité importante. Sous une simplicité apparente, la spécification de protocoles est une activité délicate qui a nécessité la recherche de formalismes de description (automates communicants, réseaux de Petri, langage LOTOS, ...) et d'outils d'aide à la validation. Certains standard existent comme par exemple, l'appel procédural à distance. Mais beaucoup de protocoles sont adaptés à une classe spécifique d'applications (par exemple, le temps réel).

On trouve ensuite des problèmes plus généraux que l'on peut classer en deux catégories : d'une part, des problèmes connus du monde parallèle (exclusion mutuelle, interblocage, atomicité, réplication par exemple) mais qui ont dû être reconsidérés et d'autre part, des problèmes nouveaux apparus avec la répartition des traitements et des données. On peut citer par exemple le problème de la terminaison d'une application répartie (problème simple en centralisé, mais délicat en réparti), le calcul d'états globaux (appelés clichés), la détection d'états stables ou inévitables, la réalisation d'un consensus<sup>3</sup>, ...

### 4.3 Les langages de programmation

À première vue, la répartition n'implique pas de grande révolution dans les langages de programmation. En effet, dans un effort minimaliste, il suffit d'utiliser une interface de programmation (API) permettant d'échanger des messages, soit deux primitives *émettre* et *recevoir* un message. L'interface à base de "prises" (sockets) offerte par le système Unix n'en propose pas beaucoup plus encore aujourd'hui.

---

<sup>3</sup>Ce problème consiste à obtenir qu'une même décision soit adoptée par tous les participants (ici processus)

On peut cependant être plus ambitieux en proposant par exemple des structures de contrôle facilitant le raisonnement et la programmation des échanges de messages. De nombreuses propositions ont été faites. Il est difficile d'en donner une liste exhaustive. Les approches gardant une communication explicite par messages, ont introduit la notion de non-déterminisme en réception voire en émission. Le non-déterminisme en réception est en effet important car l'on doit souvent exprimer l'attente d'un message parmi plusieurs possibles de type différent. Il est alors intéressant de posséder une structure de contrôle permettant d'associer à chaque type de message une action spécifique. On trouve ce type de construction dans des langages comme OCCAM et Ada.

Enfin et surtout, une extension de la notion de procédure a été proposée : l'appel procédural à distance. Le programmeur garde son modèle de programmation de type procédural, mais en fait les procédures appelées par un processus donné peuvent être localisées sur des sites distants. Ce modèle de programmation réparti est celui qui est aujourd'hui le plus utilisé et connu sous le nom de modèle client-serveur. Au niveau langage, il faut introduire un langage de définition d'interface (IDL<sup>4</sup>) qui permet de décrire l'interface des procédures qui doivent pouvoir être appelées à distance. Les problèmes d'implantation sont masqués au programmeur par une génération automatique des routines de traitement des appels côté client (appelant) et serveur (appelé) à partir des descriptions IDL.

#### 4.4 Les systèmes d'exploitation

Les systèmes d'exploitation sont évidemment les premiers concernés par la répartition. Il leur revient d'assurer l'interface avec le medium de communication.

Les concepteurs de systèmes ont envisagé deux approches possibles :

- reprendre le problème à la base et concevoir de nouveaux noyaux d'exécution répartie à partir d'une architecture matérielle. L'avantage d'une telle approche est de réduire la taille du noyau d'exécution, qualifié de micro-noyau, en limitant celui-ci aux fonctionnalités de base : gestion des ressources locales (mémoire, périphériques), gestion du parallélisme et de la communication. Les services classiques d'un système d'exploitation (tel que la gestion des fichiers) deviennent simplement des services hors du noyau d'exécution proprement-dit. La modularité du système obtenu est de ce fait beaucoup plus élevée que celle d'un système d'exploitation centralisé. L'inconvénient d'une telle approche est qu'il faut redémarrer à zéro. On peut citer une réussite française en la matière avec le système CHORUS qui est un micro-noyau d'exécution répartie orienté Temps Réel.
- étendre les systèmes d'exploitation centralisés en ajoutant au minimum une interface de communication et si possible quelques services répartis, par exemple, un système de gestion de fichiers répartis. L'avantage de cette approche est la continuité et la réutilisation. C'est ainsi par exemple que le système Unix a été progressivement étendu avec une interface de communication par sockets, par RPC<sup>5</sup>, puis par un service de fichiers répartis (NFS<sup>6</sup>).

L'inconvénient de cette approche est le caractère moins modulaire du système obtenu. Les services offerts, comme la gestion des fichiers, sont inclus dans le noyau et donc pratiquement figés.

En réalité, les deux approches se sont mutuellement fertilisées dans la mesure où les concepts introduits dans les micro-noyaux répartis ont amené à modifier la conception des systèmes classiques en essayant de leur donner une plus grande modularité.

#### 4.5 Les environnements d'exécution répartie (middleware)

Un problème de base des systèmes répartis est de maîtriser l'hétérogénéité aussi bien matérielle que logicielle (systèmes d'exploitation, langages de programmation par exemple). L'objectif est de pouvoir faire communiquer et coopérer des composants a priori hétérogènes. Pour résoudre ce problème, le modèle adopté repose sur d'une part, le schéma de communication client/serveur et d'autre part, sur la notion de "bus logiciel". Un bus logiciel doit permettre d'accéder à des services spécifiés par leur interface. Ces interfaces

---

<sup>4</sup>IDL : Interface Definition Language

<sup>5</sup>Remote Procedure Call

<sup>6</sup>Network File System

sont enregistrées dans des sortes d'annuaires qui permettent de trouver le ou les nœuds serveurs (qui auront été au préalable répertoriés).

Une norme de fait existe actuellement avec l'environnement CORBA (Common Object Request Broker) défini par l'OMG (Object Management Group Architecture). Une spécification d'un bus logiciel à objet a été définie et implantée par de nombreux développeurs. Cette couche logicielle se place donc entre le système d'exploitation et les applications.

## 5 Principes de conception

La démarche adoptée par les concepteurs de systèmes répartis repose sur un paradoxe un peu décevant. Pour caricaturer, on pourrait résumer leur principe fondamental par :

Pour concevoir un bon système réparti, concevez un système qui "a l'air centralisé" ( qui s'utilise comme )

Autrement dit, l'objectif est de faire en sorte que le système réparti ait une interface la plus proche possible de celle d'un système centralisé. Pourquoi une telle approche ?

La raison en est simple : le programmation sous les hypothèses centralisées (horloge globale et mémoire globale) est plus simple que la programmation sous les hypothèses réparties. Par conséquent, il vaut mieux "masquer" le plus possible les propriétés délicates de la répartition.

En fait, il est impossible de masquer totalement la répartition. D'une part, parce qu'il faut parfois savoir où est physiquement la ressource que l'on utilise (par exemple, une imprimante), d'autre part et surtout parce que la fiabilité du système global n'est pas suffisante. À titre d'exemple, considérons le mécanisme d'appel procédural à distance (RPC). Ce mécanisme permet d'utiliser le concept bien connu de procédure en l'étendant au contexte réparti. La procédure appelée peut être exécutée sur un site différent de celui de la procédure appelante. Cette extension permet de programmer simplement des traitements répartis mettant en jeu plusieurs nœuds en conservant le style de programmation procédural. Cependant, la fiabilité d'un appel réellement distant est bien inférieure à celle d'un appel procédural local. La mise en œuvre du mécanisme de RPC nécessite des échanges de messages, met en jeu deux sites distincts, pose des problèmes de conversion de format des données, ... Le résultat est qu'il faut que le programmeur prévoit par exemple qu'un appel peut échouer parce que le site appelé est arrêté. Pire, la détection de cette événement peut être erronée : l'appelant considère que l'appel a échoué, mais en réalité, le site appelé a exécuté la procédure.

Dès lors, quelles sont les propriétés qui peuvent masquer en partie ou totalement la répartition des données et des traitements ? On appelle de telles propriétés, des propriétés de transparence.

### 5.1 Transparence d'accès

Cette propriété consiste à assurer que l'interface d'accès à une ressource (fichier par exemple), sera identique que la ressource soit locale ou distante.

#### Exemples

- Considérons un opération d'ouverture d'un fichier sous Unix :

```
fd = open("projet/test.data",...);
```

Que le fichier spécifié soit local ou distant, le sous-système de gestion de fichiers répartis NFS permet d'utiliser la même primitive **open**. On a bien la transparence d'accès.

- Considérons une commande d'impression sous Solaris :

```
lp -d lj test.data
```

L'imprimante peut être connectée localement ou distante, la commande utilisée restera **lp** ;

- Considérons une opération de connexion à distance :

*rlogin blaise.irit.fr -l arthur*

La transparence d'accès n'est pas vérifiée puisque l'on utilise une commande différente de la commande locale.

## 5.2 Transparence de localisation

Cette propriété consiste à assurer que la désignation de la ressource est indépendante de la localisation de cette ressource. Autrement dit, les usagers peuvent ignorer la localisation réelle. Si la transparence d'accès est de plus assurée, on peut alors utiliser une ressource en ignorant si elle est locale ou distante. Cette propriété peut être affinée en assurant que tous les usagers de la ressource pourront la désigner par le même nom global indépendamment de leur propre localisation. La désignation est alors qualifiée d'uniforme.

### Exemples

- Si l'on reprend l'opération d'ouverture d'un fichier,

*fd = open("projet/test.data",...);*

NFS assure la transparence de localisation <sup>7</sup>

- Pour la commande d'impression :

*lp -d lj test.data*

l'imprimante porte un nom symbolique qui ne contient pas d'information sur sa localisation ( pour ce type de ressource, il est bien entendu utile de savoir finalement où elle se trouve);

- Pour l'opération de connexion à distance :

*rlogin blaise.irit.fr -l arthur*

la transparence de localisation n'est évidemment pas possible.

## 5.3 Transparence du partage

Cette propriété consiste à assurer que les accès concurrents à la ressource seront contrôlés de telle façon que l'intégrité de la ressource soit assurée. À titre d'exemple, pour un fichier, la transparence du partage implique de garantir les règles de synchronisation dites des lecteurs/rédacteurs. Pour une imprimante, il s'agit d'éviter les mélanges des requêtes d'impression en assurant un accès exclusif.

Les systèmes d'exploitation répartis se contentent en général d'assurer le minimum vital. Si le partage d'une imprimante est garanti sans problème (principe du spooling), le partage des fichiers est plus problématique. À titre d'exemple, le système NFS essaie de minimiser les risques d'incohérence, mais sans garantie absolue (cf. fichiers répartis). Une telle approche tient au coût de supervision du partage pour des fichiers qui, dans leur grande majorité, ne seront que des fichiers temporaires non partagés.

Ce sont donc des systèmes plus dédiés à un contexte d'application particulier qui garantissent cette transparence. Le contexte le plus répandu est celui des bases de données réparties et des sous-systèmes transactionnels associés. Les enregistrements d'une base de données sont bien des ressources fortement partagées par nature. Il est donc important d'offrir dans un contexte réparti le même type de service que celui apporté par les systèmes transactionnels centralisés. Le problème de base est celui de l'atomicité des traitements (transactions) sur la ou les bases de données.

---

<sup>7</sup>Attention, NFS n'assure pas forcément une désignation uniforme. Le même fichier sera éventuellement accessible sous des noms différents selon la localisation de l'appelant.

## 5.4 Transparence de la réplication

Cette propriété consiste à assurer que l'accès à une ressource soit identique quel que soit la forme d'implantation de cette ressource, en particulier répliquée. C'est la même idée que la transparence d'accès mais étendue à la réplication.

Ce genre de transparence est évidemment dédié à des systèmes très spécifiques. La réplication a pour objectif d'obtenir une plus grande robustesse vis-à-vis des fautes. C'est donc pour les systèmes tolérants aux fautes que la réplication est en général mise en œuvre.

## 5.5 Transparence des fautes

De façon plus générale, nous avons vu qu'un système réparti devait obligatoirement considérer les défaillances de certains de ses nœuds comme des événements probables. Il est donc important d'assurer une bonne tolérance aux fautes de l'ensemble des services d'un système réparti. À titre d'exemple, pour un système de gestion de fichiers, la défaillance d'un volume monté à distance mais momentanément non utilisé ne doit pas bloquer le fonctionnement global du service.

## 5.6 Transparence de la migration

Cette propriété consiste à assurer qu'une ressource pourra migrer d'un nœud à un autre sans que les usagers s'en aperçoive. Ce genre de problème a été en particulier étudié sur la ressource processus. En effet, ceci permet de déplacer un service dynamiquement vers un nœud moins chargé et donc de mettre en œuvre des stratégies de régulation de charge sur une architecture répartie.

## 5.7 Transparence de charge

Dans une architecture répartie, la régulation de la charge de chaque nœud peut permettre une meilleure exploitation du système global et une meilleure satisfaction des utilisateurs.

La mise en œuvre est cependant délicate car une régulation de la charge globale implique une bonne connaissance de l'état global du système. Or, cette connaissance est justement difficile à obtenir comme nous l'avons vu. Au mieux peut-on espérer avoir une observation d'un passé récent. C'est pourquoi, des algorithmes répartis doivent être conçus spécifiquement.

## 5.8 Transparence d'échelle

Nous avons vu qu'une architecture répartie est plus modulaire et adaptable dynamiquement qu'une configuration centralisée. L'ajout de nœuds ne nécessite pas l'arrêt du système. Cependant, le passage d'un système comportant une dizaine de sites à un système d'une centaine de sites n'est pas forcément transparent pour les usagers. C'est ce phénomène de changement d'échelle qu'il faut pouvoir contrôler et rendre le plus transparent possible.

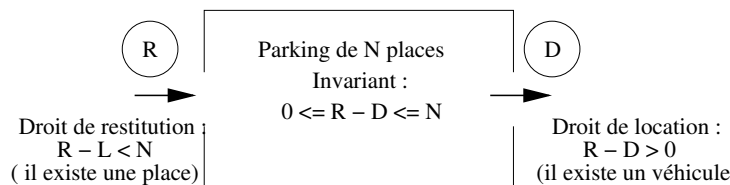


FIG. 1 – Spécification du problème

## 6 Exemple de répartition : Variations sur un thème de location

Pour bien comprendre les problèmes algorithmiques posés par la répartition, nous utiliserons un exemple que nous enrichirons progressivement. Au cours de ces variations, nous nous concentrons sur l’aspect contrôle, synchronisation du système et de ses usagers.

### 6.1 Exposition du problème

Considérons un garage de véhicules de location qui possède les propriétés suivantes :

- la capacité du garage est limitée à  $N$  places ;
- il existe deux guichets d’accès au garage : l’un que l’on peut qualifier de guichet de départ pour les prises de véhicules, l’autre que l’on peut qualifier de guichet de retour pour les restitutions. Ces deux guichets sont distants géographiquement pour faciliter la circulation des véhicules (par exemple l’un au nord du garage et l’autre au sud).
- Les véhicules présents dans le garage peuvent être loués et sortir par l’issue correspondante et réciproquement, des véhicules loués peuvent être restitués.
- Les véhicules loués ne sont pas forcément restitués à ce garage et inversement, les véhicules restitués ne sont pas obligatoirement issus de ce garage.

On notera *Restituer* et *Louer* les deux opérations qui constituent l’interface du garage. Le guichet de retour contrôle l’exécution de l’opération *Restituer*. Celui de départ contrôle l’exécution de l’opération *Louer*.

### 6.2 Le contrôle des mouvements de véhicules

#### Spécification

Les locations et restitutions des véhicules doivent donc être contrôlées de façon à garantir qu’il n’y a pas trop de véhicules dans le garage ni qu’on loue plus de véhicules qu’il n’en existe. Cette propriété de sûreté se traduit simplement sous la forme d’un invariant en introduisant deux compteurs monotones croissants : l’un comptant les véhicules restitués ( $R$ ) et l’autre les véhicules loués ( $D$ ). Leur différence doit être exactement le nombre de places occupés (le nombre de véhicules présents) dans le garage.

$$\text{invariant } 0 \leq (R - D) \leq N \tag{1}$$

La figure (1) illustre les spécifications de ce système de location.

#### Remarques

1. Il faut bien noter que les compteurs  $R$  et  $D$  n’ont pas forcément une valeur nulle initialement. Il faut simplement que leur différence ( $R - D$ ) reflète bien le nombre de véhicules présents dans le garage conformément à l’invariant.
2. Ce problème constitue un système dit réactif composé d’une partie “environnement” représentée par les véhicules et d’une partie “contrôle” assurant la gestion correcte de la ressource garage.
3. L’invariant (1) doit être vérifié par le système global composé des véhicules et des deux issues guichets.

Indépendamment de tout problème de répartition, l'entrée ou le départ d'un véhicule est donc soumis à précondition.

$$\begin{aligned} \text{Pré}(\textit{Restituer}) & : (R - D) < N \\ \text{Pré}(\textit{Louer}) & : 0 < (R - D) \end{aligned}$$

Compte tenu des hypothèses, la connaissance du compteur des entrées  $R$  est localisée au guichet de restitution. Celle du compteur des départs  $D$  est localisée au guichet de départ. Or, on peut constater que le guichet de retour, comme celui de départ, doit avoir une connaissance "globale" des deux compteurs pour pouvoir décider si une opération est acceptable en évaluant la précondition correspondante.

Autrement dit, nous sommes en face d'un problème réparti. Comment chaque guichet peut-il avoir une connaissance suffisante du compteur géré par l'autre pour pouvoir assurer un contrôle correct sachant qu'ils ne peuvent que s'échanger des messages ? Nous allons envisager plusieurs possibilités de répartition du contrôle.

### 6.2.1 Répartition par accès distant (maintien d'un chef d'orchestre)

Une version simple consiste à "centraliser" les variables d'états  $R$  et  $D$  de façon à pouvoir évaluer les deux prédicats. Il faut donc introduire un troisième nœud "central" qui implante un service **Contrôle** accessible à distance offrant les deux opérations :

```
interface Contrôle() {
    IncrR() /* incrémenter le compteur R */
    IncrD() /* incrémenter le compteur D */
}
```

Sur le nœud central, il faudra donc démarrer un serveur acceptant l'interface précédente. Ce serveur devra assurer le maintien de l'invariant (1) par une synchronisation correcte des deux opérations. Le guichet de restitution exécute une opération locale de restitution en commençant par appeler la procédure **IncrR** à distance. La terminaison de cet appel autorise le guichet à continuer son opération locale de restitution d'un véhicule. Il en est de même pour le guichet de départ avec l'opération **IncrD**.

L'inconvénient de cette solution est qu'elle nécessite un appel procédural à distance pour chaque opération de restitution ou de location. De plus, la disponibilité du service repose sur le bon fonctionnement des trois nœuds.

### 6.2.2 Répartition par image (deux solistes mais ils s'observent)

Supposons que, par un protocole adéquat à définir, chaque guichet puisse avoir une "image" du compteur distant (source) de l'autre. La sémantique d'un compteur image peut être définie informellement par les propriétés de sûreté et de vivacité suivantes :

- le compteur image a initialement la même valeur que le compteur "source" ;
- le compteur image ne peut prendre que des valeurs prises par le compteur source et ceci dans le même ordre chronologique ;
- le compteur image peut ne pas prendre toutes les valeurs successives du compteur source ;
- si le compteur source prend la valeur  $k$ , le compteur image finira par prendre une valeur égale ou supérieure à  $k$ .

On notera cette relation dite d'observation entre une source et une image <sup>8</sup> par le symbole  $\prec$ . Pour le garage, on peut donc supposer que les relations d'observations suivantes vont être mises en œuvre :

$$'R \prec R \wedge 'D \prec D$$

Compte tenu des propriétés énoncées sur les observations, l'image d'un compteur croissant est toujours inférieure ou égale à la valeur de sa source (Petit théorème facile à démontrer). On a donc :

$$\text{invariant } 'R \leq R \wedge 'D \leq D \tag{2}$$

---

<sup>8</sup>Cette notion d'image est similaire à la notion de cache en cohérence faible

On a par conséquent le théorème suivant :

$$R \geq D \wedge 'D \leq D \vdash (R - 'D < N \Rightarrow R - D < N)$$

Autrement dit, le guichet de restitution, s'il possède localement l'image ' $D$ , peut décider à coup sûr en évaluant le prédicat **local** mais approché  $R - 'D < N$ . Il en est de même pour le guichet de location avec :

$$R \geq D \wedge 'R \leq R \vdash ('R - D > 0 \Rightarrow R - D > 0)$$

Une solution au problème consiste donc ici à “localiser” l'évaluation des préconditions réparties grâce à l'utilisation d'images. Il suffira d'implanter le mécanisme d'observation. Ceci peut être fait de multiple façons. Par exemple, si l'on suppose que les deux guichets sont reliés par un canal de communication bidirectionnel respectant l'ordre chronologique des envois de messages, le mécanisme d'observation peut être implanté par :

- l'envoi périodique ou à chaque modification de la valeur de la source vers le nœud de l'image ;
- l'appel procédural à distance, issu de la source, de l'opération *image.incr(1)* par la source répliquant chaque opération locale *source.incr(1)* ;
- l'appel procédural à distance, issu de l'image, de l'opération *source.lire* pour obtenir la valeur de la source, appel qui peut être périodique ou à la demande lorsque le prédicat à évaluer est faux par exemple ;
- etc, etc , ...

Cet exemple montre l'absence d'état global nécessitant ici d'utiliser une connaissance approchée impliquant la condition globale attendue.

Les avantages que l'on peut tirer de la répartition du contrôle dans cette solution sont variés :

- minimisation possible des échanges de messages. Le protocole d'observation peut s'ajuster aux besoins applicatifs ;
- fiabilité du système : il y a un composant en moins ;
- disponibilité : même si un des deux guichets connaît une défaillance momentanée, l'autre peut assurer encore son rôle dans la mesure où la précondition reste valide avec la dernière valeur d'image obtenue.

Sur ce dernier point, le redémarrage du guichet défaillant pose cependant le problème de la restauration d'une valeur correcte du compteur associé. Il faut assurer que la valeur courante des compteurs de chaque guichet est robuste, c'est-à-dire qu'elle survit à un arrêt.

### 6.2.3 Répartition par circulation (une partition nomade)

L'idée de base de cette solution est que l'on peut “simuler” des variables globales en faisant circuler équitablement entre les nœuds un message qui contient la valeur courante de ces variables. On parle aussi de jeton circulant valué. Lorsqu'un nœud possède ce jeton ( $\equiv$  a reçu le message unique qui circule), il peut considérer qu'il possède le droit d'utilisation ainsi que la valeur courante de la variable globale.

Dans notre problème, il suffit de faire circuler entre les deux guichets un message “jeton” unique valué par la différence des deux compteurs  $VP = R - D$ . Cette différence représente, ne l'oublions pas, le nombre de véhicules présents dans le garage. Le guichet de restitution incrémente  $VP$  et le guichet de départ le décrémente en garantissant l'invariant : **invariant**  $0 \leq VP \leq N$ .

Les opérations de restitution et de location sont alors conditionnées non seulement par l'invariant précédent, mais aussi par la présence du jeton. À titre d'exemple, l'algorithme d'une opération **Restituer** utilisant le jeton valué sera de la forme suivante :

```

restituer() {
    int vpl;
    jeton.obtenir();
    vpl = jeton.val(); vpl++; jeton.affecter(vpl);
    jeton.libérer;
}

```

**Remarque** Ne pas oublier qu'il faut assurer la traversée du nœud par le jeton lorsqu'il n'existe aucune requête en attente localement.

La circulation du jeton se fait en général sur un anneau logique passant par tous les nœuds. Un problème non négligeable de cette approche est donc le bruit de fond provoqué par la circulation permanente du message jeton même si aucune opération n'est en attente sur un nœud. Un deuxième problème se pose avec le risque de perte du jeton en cas de défaillance dans les communications (perte du message jeton) ou de défaillance d'un nœud (arrêt du nœud).

### 6.3 Enrichissement du système (Plusieurs orchestres)

Supposons maintenant qu'il existe plusieurs compagnies de location (Hertz, Ada, Europcar, ...) et qu'elles aient décidé de partager le garage sans pour autant mettre en commun leurs systèmes d'information. Chaque compagnie possède donc un guichet de retour et un de départ et ne loue que ses propres véhicules. Si l'on suppose qu'il existe  $P$  compagnies, on a donc  $P$  compteurs  $D_i$  et  $P$  compteurs  $R_i$  répartis dans chaque guichet.

#### 6.3.1 Répartition par observation et circulation

Pour un guichet de départ  $i$ , la précondition à évaluer est :

$$\text{Pré}(\text{Louer}) \quad : \quad (R_i - D_i) > 0$$

Cette condition indique que la compagnie  $i$  possède encore au moins un véhicule à louer dans le garage. Une solution simple pour une évaluation locale au guichet de départ, est d'utiliser des observations. Le guichet de départ doit posséder une image ' $R_i$  du compteur  $R_i$ . On aura la propriété :

$$('R_i - D_i > 0) \Rightarrow (R_i - D_i > 0)$$

Pour un guichet de retour, le problème est un peu plus difficile. Il faut en effet tester la précondition :

$$\text{Pré}(\text{Restituer}) \quad : \quad (\sum_{i=1}^{i=P} R_i - \sum_{i=1}^{i=P} D_i) < N$$

Pour les compteurs  $D_i$ , chaque guichet de retour peut se contenter d'une approximation. On peut donc placer sur un guichet de retour  $i_0$ , les images de tous les autres compteurs  $D_j$ ,  $j \neq i_0$ <sup>9</sup>.

Pour un guichet  $i_0$ , on aura donc besoin des relations d'observation  $\langle \wedge j = 1, P : 'D_j^{i_0} \prec D_j \rangle$  et, par suite, l'implication suivante sera satisfaite :

$$(\sum_{i=1}^{i=P} R_i - \sum_{j=1}^{j=P} 'D_j^{i_0}) < N \Rightarrow (\sum_{i=1}^{i=P} R_i - \sum_{i=1}^{i=P} D_i) < N$$

Reste le problème de la somme des compteurs  $R_i$ . Il faut pouvoir disposer de la valeur exacte ou d'une approximation par excès. Une solution possible est celle du jeton. Une variable globale abstraite  $S_R$  peut être représentée par un jeton circulant entre les guichets de retour placés sur un anneau logique. Pour préciser la localisation de l'objet (de la variable) jeton mobile, on introduit le prédicat  $X@node(i)$ . Ce prédicat est vrai si la variable  $X$  est localisée sur le site  $node(i)$ . Le test devient alors possible sous la forme :

$$S_R@node(i_0) \wedge (S_R - \sum_{j=1}^{j=P} 'D_j^{i_0}) < N$$

---

<sup>9</sup>Pour simplifier l'écriture des formules, on suppose que sur le site  $i_0$ , l'image ' $D_{i_0}$  est un alias pour désigner la source  $D_{i_0}$  puisque image et source sont sur le même site

### 6.3.2 Répartition par réplication

Une autre solution consiste à répliquer la variable somme  $S_R$  sur tous les guichets de retour. Il se pose alors le problème de la cohérence globale de ces copies. Il est nécessaire de respecter un protocole bien précis pour assurer une mise à jour dite “atomique” des différentes copies<sup>10</sup>. À titre d'exemple des problèmes qui peuvent survenir, il suffit de considérer le cas de deux guichets de retour  $G_{R1}$  et  $G_{R2}$ . Supposons que l'on ait donc deux copies  $S_{R1}$  et  $S_{R2}$ . On peut envisager le maintien de la cohérence des deux copies en répliquant tout simplement chaque opération logique sur les deux copies. Ainsi, une opération de restitution ayant pour origine  $G_{R1}$  devra se traduire par une opération locale  $S_{R1}.incr(1)$  et une opération distante  $S_{R2}.incr(1)$ . Symétriquement pour une opération de restitution ayant pour origine  $G_{R2}$ , il faudra exécuter une opération locale  $S_{R2}.incr(1)$  et une opération distante  $S_{R1}.incr(1)$ . Supposons le déroulement chronologique suivant :

Restitution ayant pour origine le guichet	$G_{R1}$	$G_{R2}$
Opération locale	$S_{R1}.incr(1)$	
“		$S_{R2}.incr(1)$
Opération distante		$S_{R1}.incr(1)$
“	$S_{R2}.incr(1)$	

Si la valeur des deux copies était cohérente avant ces deux opérations de restitution mais que la précondition  $(S_{R_i} - \sum_{j=1}^{j=P} D_j^{i_0}) < N$  est devenue fausse aussi bien pour  $R_1$  que pour  $R_2$  après les deux incrémentations locales, les deux opérations distantes deviennent impossibles. En fait, les deux opérations de restitution ont été partiellement exécutées. Pourtant, il était possible d'exécuter complètement une SEULE des deux restitutions. Il ne fallait simplement pas entrelacer les opérations d'incrémentations. Des protocoles de diffusion atomiques ont été proposés pour résoudre ce genre de situation.

## 6.4 Conclusion

Nous avons vu à travers cet exemple, à la fois les difficultés et la richesse des solutions réparties. Encore, n'avons nous pas considéré le système d'information proprement dit. En effet, chaque guichet devra sans doute gérer un journal des opérations qu'il a traité par exemple dans une journée. Ces journaux enregistrés sur chaque guichet seront éventuellement transmis à un site central chaque nuit durant la fermeture du garage. Inversement, un site central (isateur) pourra interroger régulièrement les guichets.

On peut aussi envisager une régulation entre garages s'il en existe plusieurs implantés dans différentes villes. Il faudra alors définir un protocole d'échange permettant d'éviter à la fois les “ruptures de stock” et les engorgements des garages.

Dans tous les cas, certains problèmes récurrents devront être résolus : choix de la répartition des données, synchronisation globale, atomicité des opérations, réplication, diffusion, . . .

## 6.5 Exercices

1. Un jeton se perd. Imaginer des solutions pour détecter cette perte et engendrer un nouveau jeton (Attention, il doit être unique!).
2. La circulation permanente du jeton consomme de la bande passante réseau inutilement. Imaginer un protocole permettant d'arrêter le jeton sur un site et de le remettre en mouvement lorsqu'une requête apparaît sur un site quelconque.
3. Réfléchir au problème posé par l'arrêt d'un guichet de retour, de départ notamment vis-à-vis des observations.
4. Concevoir une régulation possible entre plusieurs garages de location.
5. Imaginer un protocole qui éviterait le problème des entrelacements erronés d'opérations répliquées.

---

<sup>10</sup>L'atomicité est une propriété qui n'est pas spécifique aux systèmes réparties. Les systèmes transactionnels (centralisés) ont largement étudié ce problème. Cependant, la répartition pose des problèmes de mise en œuvre plus complexes.