République Algérienne Démocratique et Populaire

UNIVERSITE MUSTAPHA STAMBOULI DE MASCARA

Faculté Des Sciences Exactes

Plolycopié de Cours

Fondements des Systèmes Répartis (FSR)

Dr. Sabrina ABID

Ce cours est destiné aux étudiants de Master spécialité Réseaux et systèmes distribués (RSD)

AVANT PROPOS

Ce cours est destiné aux étudiants de première année Master informatique spécialité Réseaux et systèmes distribués.
Connaissances préalables recommandées : Des connaissances préliminaires sur les systèmes d'exploitation vus en licence.
Ce module s'inscrit dans la continuité du module système d'exploitation 2 vus en Licence et a pour but d'encourager les étudiants à détecter des défis des systèmes distribués, à préparer et à présenter des exposés pour montrer ces défis.
Les étudiants sont invités à présenter les approches existantes pour résoudre tels défis et à proposer des solutions.
Charge horarire : 2 séances de cours par semaines (3h), une séance de TD (1h30).
Dans un premier temps, on détaille les concepts généraux des systèmes distribués en introduisant l'historique des systèmes informatiques et leurs classification, les domaine d'applications des systèmes distribués. On explique la notion d'absence d'horloge global et d'état global pour introduire la datation logique dans les systèmes distribué et la construction d'état global pour ensuite entamer les coupures. On passe à la définition de la réplication et ses objectifs d'applications. Finalement on expose deux méthodes de tolérance aux pannes dans les systèmes distribués.

Février 2021

Table des matières

Introduction générale

Le domaine de l'informatique distribuée ou des Systèmes Répartis (SR) couvre tous les aspects de l'informatique et de l'accès à l'informations à travers plusieurs éléments de traitement connectés par toute forme de réseau de communication, qu'il soit local ou étendu dans la couverture.

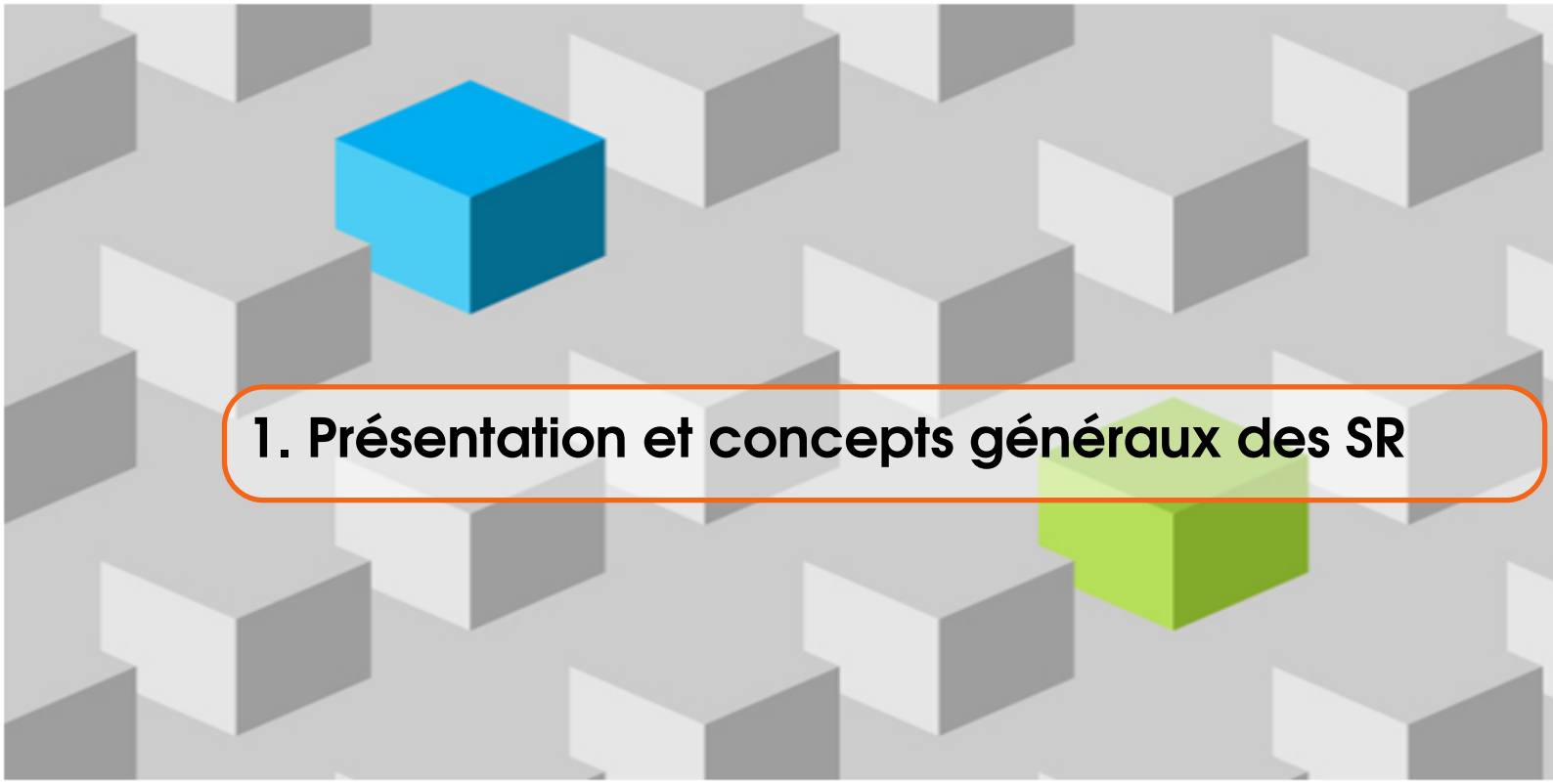
Depuis l'avènement d'Internet dans les années 1970, il y a eu une croissance régulière de nouvelles applications nécessitant un traitement distribué. Cela a été activé par les progrès de la technologie des réseaux et du matériel, la baisse du coût du matériel, et une plus grande sensibilisation des utilisateurs finaux. Ces facteurs ont contribué à faire de l'informatique distribuée une solution rentable, performante et tolérante aux pannes.

Au tournant du millénaire, il y a eu une croissance explosive de l'expansion et de l'efficacité d'Internet, qui a été couplé par un accès accru aux ressources en réseau via le World Wide Web, partout dans le monde. Couplé à une croissance tout aussi spectaculaire du sans fil et les zones de réseaux mobiles, et la chute des prix de la bande passante et périphériques de stockage, nous assistons à une poussée rapide des applications distribuées et un intérêt connexe pour le domaine de l'informatique distribuée dans les universités, les organisations gouvernementales et les institutions privées.

Les progrès de la technologie matérielle ont soudainement fait de la mise en réseau des capteurs une réalité, et les réseaux embarqués et de capteurs deviennent rapidement une partie intégrante de la vie de chacun - du réseau domestique aux gadgets à l'automobile communiquant par GPS (Global Positioning System, système de positionnement global), au bureau entièrement en réseau avec surveillance. Global Positioning SystemGlobal Positioning System

Clairement, c'est un domaine très important. De plus, ce domaine en évolution se caractérise par un large éventail de défis pour lesquels les solutions doivent avoir des fondements sur des principes solides.

Le domaine de l'informatique distribuée est très important, et il y a une énorme demande pour un bon polycopié complet. Ce cours couvre en détail tous les sujets importants en profondeur, en combinant cela avec une clarté d'explication et une facilité de compréhension. Le cours est organisé en six chapitres et sera particulièrement précieux pour les communauté universitaire et l'industrie informatique en général.



1. Présentation et concepts généraux des SR

1.1 Introduction

Les réseaux d'ordinateurs sont partout. Internet en est un, tout comme les nombreux réseaux dont il est composé. Réseaux de téléphonie mobile, réseaux d'entreprise, réseaux d'usine, réseaux de campus, réseaux domestiques, réseaux embarqués - tout cela, à la fois séparément et en combinaison, partagent les caractéristiques essentielles qui en font des sujets pertinents pour étude sous la rubrique systèmes distribués. Dans ce chapitre, nous définissons un système distribué comme un système dans lequel des composants matériels ou logiciels situés sur des ordinateurs en réseau communiquent et coordonnent leurs actions uniquement par des messages. Cette définition simple couvre toute la gamme des systèmes dans lesquels les ordinateurs en réseau peuvent être utilement déployés.

Les ordinateurs connectés par un réseau peuvent être spatialement séparés par toute distance. Ils peuvent être sur des continents séparés, dans le même bâtiment ou dans la même pièce.

Notre définition des systèmes distribués a les conséquences importantes suivantes:

— **Concurrence:** dans un réseau d'ordinateurs, l'exécution de programmes concurrent est la norme. Je peux faire mon travail sur mon ordinateur pendant que vous faites votre travail sur le vôtre, en partageant des ressources telles que des pages Web ou des fichiers si nécessaire. La capacité du système à gérer les ressources partagées peut être augmenté en ajoutant plus de ressources (par exemple. ordinateurs) au réseau.

— **Pas d'horloge globale:** lorsque les programmes doivent coopérer, ils coordonnent leurs actions en échangeant des messages. Une coordination étroite dépend souvent d'une idée partagée du temps au cours de laquelle les actions des programmes se produisent. Mais il s'avère qu'il y a des limites à la précision avec laquelle les ordinateurs d'un réseau peuvent synchroniser leurs horloges - là Il n'y a pas de notion globale unique de temps correcte. Ceci est une conséquence directe du fait que la seule communication consiste à envoyer des messages via un réseau.

— **Pannes indépendantes:** tous les systèmes informatiques peuvent tomber en panne, et c'est la responsabilité des concepteurs de systèmes qui doivent prévoir les conséquences d'éventuelles pannes. Les systèmes distribués peuvent échouer de plusieurs manières. Les pannes du réseau entraînent l'isolement des ordinateurs qui sont connectés, mais cela ne signifie pas qu'ils

cessent de fonctionner. En fait, les programmes sur ces ordinateurs peuvent ne pas être en mesure de détecter si le réseau est en panne ou est devenu inhabituellement lent. De même, la panne d'un ordinateur ou l'arrêt inopiné d'un programme quelque part dans le système (un plantage), n'est pas immédiatement signalé au autres composants avec lesquels il communique.

La motivation première pour construire et utiliser des systèmes distribués provient d'un désir pour partager des ressources. Le terme «ressource» est plutôt abstrait, mais il caractérise le mieux la gamme de choses qui peuvent être utilement partagées dans un système informatique en réseau. Il s'étend des composants matériels tels que les disques et les imprimantes aux logiciels définis des entités telles que des fichiers, des bases de données et des objets de données de toutes sortes.

Le but de ce chapitre est de donner une vision claire de la nature des systèmes distribués et les défis qui doivent être relevés pour s'assurer qu'ils sont réussi.

La section 1.2 donne une définition des systèmes distribués selon plusieurs auteurs, objectif et motivations de ces système et quelques exemples illustratifs de systèmes distribués, tandis que la section 1.3 décrit les défis auxquels sont confrontés les concepteurs de systèmes distribués: hétérogénéité, ouverture, sécurité, évolutivité, gestion des pannes, concurrence d'accès, transparence et qualité de service. La section 1.4 présente une comparaison entre les systèmes centralisés et les systèmes distribués.

1.1.1 Historique

- 1945-1980 : l'apparition du premier ordinateur (Mainframe) grosse machine qui pesaient très lourds, encombrait beaucoup de place et traitait peu d'instruction par seconde.
- A partir de 1985: il y a eu deux avancées technologiques:
  1. Réseaux de communication: (LAN, WAN,...,internet)
  2. Développement de puissance de calcul des machines qui exécutent plusieurs instructions par secondes, on est passée des microprocesseurs 8 bits, 16, 32 à 64 bits.
- Évolution des besoins des utilisateurs qui deviennent de plus en plus gourmand en communication (réseaux sociaux, ...),
- les besoins de communication à distance des entreprise (e_commerce, e_learning, e_gov, ...)
- le besoin de collaboration à distance
- Aujourd'hui on dispose de beaucoup plus de ressources (de la voix, du texte, des données ....).

  En conséquence, on à une convergence entre l'évolution de la technologie d'une part et des besoins des utilisateurs d'autre part, d'où on est passé vers les systèmes informatiques qui au lieu d'être composé d'une seule machine sont composés de plusieurs machines connectées en réseau.

1.1.2 Classification des systèmes informatiques d'après Flynn

La taxonomie de Flynn est une classification des architectures d'ordinateur, proposée par Michael Flynn en 1961 [Fly72], les quatre catégories définies par Flynn sont classées selon le type d'organisation du flux de données et du flux d'instructions.

- SISD (unique flux d'instructions, unique flux de données) Il s'agit d'un ordinateur séquentiel qui n'exploite aucun parallélisme, tant au niveau des instructions qu'au niveau de la mémoire. Cette catégorie correspond à l'architecture de von Neumann.
- SIMD (unique flux d'instructions, multiples flux de données) Il s'agit d'un ordinateur qui utilise le parallélisme au niveau de la mémoire, par exemple le processeur vectoriel.
- MISD (multiples flux d'instructions, unique flux de données) Il s'agit d'un ordinateur dans lequel une même donnée est traitée par plusieurs unités de calcul en parallèle. Il existe peu d'implémentions en pratique. Cette catégorie peut être utilisée dans le filtrage numérique et

la vérification de redondance dans les systèmes critiques.

— MIMD (multiples flux d'instructions, multiples flux de données) Dans ce cas, plusieurs unités de calcul traitent des données différentes, car chacune d'elles possède une mémoire distincte. Il s'agit de l'architecture parallèle la plus utilisée, dont les deux principales variantes rencontrées sont les suivantes :
  — MIMD à mémoire partagée (fortement couplées) Les unités de calcul ont accès à la mémoire comme un espace d'adressage global. Tout changement dans une case mémoire est vu par les autres unités de calcul. La communication entre les unités de calcul est effectuée via la mémoire globale.
  — MIMD à mémoire distribuée (faiblement couplées) Chaque unité de calcul possède sa propre mémoire et son propre système d'exploitation. Ce second cas de figure nécessite un middleware pour la synchronisation et la communication.
  en fait ces systèmes qui font l'objet de notre cours qui sont les **systèmes distribués**.
  Un système MIMD hybride est l'architecture la plus utilisée par les superordinateurs. Ces systèmes hybrides possèdent l'avantage d'être très extensibles, performants et à faible coût.

1.2 Définition, objectifs, motivations, Exemples d'applications

1.2.1 Définition d'un système distribué

Diverses définitions des systèmes distribués ont été données dans la littérature, aucune d'elles n'est satisfaisante, et aucune d'entre elles n'est en accord avec les autres.

Définition 1.2.1 Un ensemble de machines connectées par un réseau, et équipées d'un logiciel dédié à la coordination des activités du système ainsi qu'au partage de ses ressources[Ca12]

Définition 1.2.2 Un système distribué est un ensemble d'ordinateurs indépendants qui apparaît à un utilisateur comme un système unique et cohérent[Ta07].

Cette définition comporte plusieurs aspects importants. Le premier est qu'un système distribué est constitué de composants (c'est-à-dire d'ordinateurs) qui sont autonomes. Le second aspect est que les utilisateurs (qu'ils soient des personnes ou des programmes) pensent qu'ils traitent avec un seul système. Cela signifie que d'une manière ou d'une autre les composants autonomes ont besoin de collaborer.

Notez qu'aucune hypothèse n'est faite concernant le type des ordinateurs. En principe, même au sein d'un même système, ils peuvent aller d'ordinateurs centraux de hautes performances vers de petits noeuds dans des réseaux de capteurs. Également, aucune hypothèse n'est faite sur la manière dont les ordinateurs sont inter-connectés.

Au lieu d'aller plus loin dans les définitions, il est peut-être plus utile de se concentrer sur les caractéristiques importantes des systèmes distribués. Une caractéristique importante est que les différences entre les ordinateurs et la manière dont ils communiquent sont généralement cachés aux utilisateurs. La même chose pour l'organisation interne du système distribué. Une autre caractéristique importante est que les utilisateurs et les applications peuvent interagir avec un système distribué de manière cohérente et uniforme, peu importe où et quand l'interaction a lieu.

En principe, les systèmes distribués devraient également être relativement faciles à étendre ou à mettre à l'échelle. échelle. Un système distribué sera normalement disponible en permanence, bien que certaines parties soient peut-être temporairement hors d'usage. Les utilisateurs et les applications ne doivent pas remarquer que des parties sont en cours de remplacement ou de réparation, ou que de nouvelles parties ajoutées pour servir plus d'utilisateurs ou d'applications [Ta07].

En résumé:

Définition 1.2.3 un système distribué (SD) est un ensemble d'entités autonomes de calcul(ordinateurs, PDA, processus, capteurs ...) interconnéctées via un réseau de communication et qui peuvent communiquer, ces machines travaillent en collaboration pour une tâche précise.

Un **SD** est l'art et la manière de faire croire qu'un ensemble de machines de loin se comporte comme une seule machine.

Caractéristiques des ressources

- Pas de partage de mémoire (pas de mémoire commune)
- Pas d'horloge globale
- Pas de possibilité d'avoir un état global instantané du système.

1.2.2 objectifs

- Utiliser et partager des ressources distantes
  - Système de fichiers : utiliser ses fichiers à partir de n'importe quelle machine
  - Imprimante : partagée entre toutes les machines
- Optimiser l'utilisation des ressources disponibles: Calculs scientifiques distribués sur un ensemble de machines
- Système plus robuste
  - réplication pour fiabilité : deux serveurs de fichiers dupliqués, avec sauvegarde
  - Plusieurs éléments identiques pour résister à la montée en charge.
- Aspects économiques: Coût : plusieurs processeurs à bas prix
- Réalisation de systèmes à haute disponibilité
- Puissance de calcul : aucune machine centralisée ne peut réaliser le puissance de calcul de plusieurs machines.
- Performance : calcul parallèle
- Adaptation : à des classes d'applications réelles naturellement distribuées
- Fiabilité : résistance aux pannes logicielles ou matérielles
- Extensibilité : croissance progressive selon le besoin

Inconvénients/ points faibles

- Si problème au niveau du réseau
  - Le système marche mal ou plus du tout
- Bien souvent, un élément est central au fonctionnement du système : serveur
  - Si le serveur plante : plus rien ne fonctionne
  - Goulet e potentiel d'étranglement si le débit d'information est très important
  - Sans élément central
    - Gestion du système totalement décentralisée et distribuée
    - Nécessite la mise en place d'algorithmes +/- complexes

1.2.3 motivations

La motivation pour utiliser un système distribué fait partie des exigences suivantes[Aja08] :

1. **Calculs intrinsèquement distribués** Dans de nombreuses applications telles que transfert d'argent dans le secteur bancaire, ou parvenir à un consensus entre les parties géographiquement éloigné, le calcul est intrinsèquement distribué.
2. **Partage des ressources** Ressources telles que périphériques, ensembles de données complets dans les bases de données, les bibliothèques spéciales, ainsi que les données (variables / fichiers) ne peuvent pas être entièrement répliquées sur tous les sites car ce n'est souvent ni pratique ni rentable. De plus, ils ne peuvent pas être placés sur un seul site car l'accès à ce site pourrait s'avérer être un goulot d'étranglement. Par conséquent, ces ressources sont généralement distribué à travers le système. Par exemple, des bases de données distribuées

comme DB2 partitionne les ensembles de données sur plusieurs serveurs, en plus de les reproduire sur quelques sites pour un accès rapide ainsi que pour la fiabilité.

3. **Accès aux données et ressources géographiquement distantes** Dans de nombreux scénarios, les données ne peuvent pas être répliquées sur chaque site participant à l'exécution distribuée car elles peuvent être trop volumineuses ou trop sensibles pour être répliqué. Par exemple, les données de paie au sein d'une multinationale sont à la fois trop volumineuses et trop sensibles pour être répliquées dans chaque succursale / site. Il est donc stocké sur un serveur central qui peut être interrogé par les succursales. De même, les ressources spéciales telles que les supercalculateurs n'existent que dans certains emplacements, et pour accéder à ces supercalculateurs, les utilisateurs doivent se connecter à distance.
   Progrès dans la conception d'appareils mobiles aux ressources limitées également comme dans la technologie sans fil avec laquelle ces appareils communiquent ont donné un nouvel élan à l'importance des protocoles distribués et middleware.
4. **Fiabilité accrue** Un système réparti a le potentiel inhérent pour fournir une fiabilité accrue en raison de la possibilité de répliquer ressources et exécutions, ainsi que la réalité que les ressources réparties géographiquement ne risquent pas de planter / de mal fonctionner en même temps dans des circonstances normales.
   La fiabilité comporte plusieurs aspects:
   — la disponibilité, c'est-à-dire que la ressource doit être accessible à tout moment;
   — l'intégrité, c'est-à-dire que la valeur / l'état de la ressource doit être correct, en face d'accès simultané à partir de plusieurs processeurs, selon la sémantique attendue par l'application;
   — la tolérance aux pannes, c'est-à-dire la capacité à se remettre des pannes du système (voir chapitre 6).
5. **Augmentation du rapport performances / coût** Par le partage des ressources et l'accès données et ressources géographiquement distantes, le rapport performances / coût est augmenté.
   Bien qu'un débit plus élevé n'ait pas nécessairement été le principal objectif derrière l'utilisation d'un système distribué, néanmoins, toute tâche peut être partitionné sur les différents ordinateurs du système distribué.
   En plus de répondre aux exigences ci-dessus, un système distribué offre également les avantages suivants:
6. **Évolutivité/scalabilité** Comme les processeurs sont généralement connectés par un réseau étendu, l'ajout de processeurs supplémentaires ne pose pas de goulot d'étranglement direct pour le réseau de communication.
7. **Modularité et extensibilité incrémentielle** Processeurs hétérogènes peuvent être facilement ajoutés au système sans affecter les performances, tant que ces processeurs exécutent les mêmes algorithmes middleware.
   De même, les processeurs existants peuvent être facilement remplacés par d'autres processeurs.

1.2.4 exemples d'application

Le but de cette section est de fournir des exemples motivants de systèmes distribués contemporains illustrant à la fois le rôle omniprésent des systèmes distribués et la grande diversité des applications associées.

Comme mentionné dans l'introduction, les réseaux sont partout et sous-tendent de nombreux services quotidiens que nous tenons désormais pour acquis: Internet et le World Wide Web associé, recherche sur le Web, jeux en ligne, e-mails, réseaux sociaux, commerce électronique, etc.

illustrer davantage ce point, considérons le tableau 1.1, qui décrit une gamme sélectionnée

de secteurs d'application clés commerciaux ou sociaux clés mettant en évidence les utilisations émergentes de la technologie des systèmes distribués.

Comme on peut le voir, les systèmes distribués englobent un bon nombre des développements technologiques les plus importants de ces dernières années et donc une compréhension de la technologie sous-jacente est absolument essentielle à la connaissance de l'informatique moderne.

le tableau 1.1 donne également un premier aperçu du large éventail d'applications utilisées aujourd'hui, des systèmes relativement localisés (tels que trouvés, par exemple, dans une voiture ou un avion) à l'échelle mondiale des systèmes impliquant des millions de nœuds, des services centrés sur les données aux taches de traitements intensifs, des systèmes construits à partir de capteurs très petits et relativement primitifs à ceux incorporant des éléments de calcul puissants, des systèmes embarqués et ainsi de suite.

Nous examinons maintenant des exemples plus spécifiques de systèmes distribués pour illustrer davantage la diversité et même la complexité de systèmes distribués.

Massively multiplayer online games (MMOGs)

Les jeux en ligne massivement multijoueurs offrent une expérience immersive grâce à laquelle de nombreux utilisateurs interagissent via Internet avec un monde virtuel persistant. Des exemples de tels jeux incluent EverQuest II de Sony et EVE Online de Finnish company CCP Games. Ces mondes ont considérablement augmenté en sophistication et incluent désormais des arènes de jeu complexes (par exemple EVE, Online se compose d'un univers avec plus de 5 000 systèmes stellaires) et de multiples systèmes sociaux et économiques. Le nombre de joueurs augmente également, avec des systèmes capables de prendre en charge plus de simultanément 50000 joueurs en ligne (et le nombre total de joueurs peut-être dix fois ce chiffre).

L'ingénierie des MMOGs représente un enjeu majeur pour la technologie des systèmes distribués, notamment en raison de la nécessité de temps de réponse rapides pour préserver l'expérience utilisateur du jeu.

Commerce financier

Comme autre exemple, nous examinons la prise en charge des systèmes distribués pour les marchés financiers.

L'industrie financière est depuis longtemps à la pointe des technologies des systèmes distribués avec son besoin, en particulier, d'accéder en temps réel à un large éventail de sources d'informations (par exemple, cours de bourse actuels, développements politiques et économiques). L'industrie utilise des applications de surveillance automatisés et de commerce.

Notez que dans ces systèmes, l'accent est mis sur la communication et le traitement des éléments d'intérêt, appelés événements dans les systèmes distribués, avec la nécessité également de fournir de manière fiable les événements et en temps opportun à un nombre potentiellement très important de clients qui ont un intérêt déclaré pour ces éléments d'information. Des exemples de tels événements incluent une baisse dans un cours de bourse, la publication des derniers chiffres du chômage, etc.

Cela nécessite un style d'architecture sous-jacente très différent des styles mentionnés ci-dessus (par exemple client-serveur), et de tels systèmes emploient généralement ce que l'on appelle systèmes distribués basés sur les événements[Ca12].

Serveur de fichiers

- Accès aux fichiers de l'utilisateur quelque soit la machine utilisée
- Machines du département informatique
  - Clients : scinfeXXX
  - Un serveur de fichier
  - Sur toutes les machines : /home/durand est le « home directory » de l'utilisateur durand
  - Physiquement : fichiers se trouvent uniquement sur le serveur

Finance et commerce	La croissance du commerce électronique, illustrée par des entreprises telles que Amazon et eBay, et les technologies de paiement sous-jacentes telles que Pay Pal; l'émergence associée de la banque et du commerce en ligne et également des systèmes complexes de diffusion d'informations pour les marchés financiers.
La société de l'information	La croissance du World Wide Web en tant que entrepôt d'informations et connaissance; le développement de moteurs de recherche Web comme Google et Yahoo pour rechercher dans ce vaste entrepôt; l'émergence des bibliothèques numériques et la numérisation à grande échelle des sources d'information existantes comme des livres (par exemple, Google Books); l'augmentation importante du contenu généré par les utilisateurs via des sites tels que YouTube, Wikipédia et Flickr; l'émergence du réseau social à travers des services tels que Facebook et MySpace.
Industries créatives et divertissement	L'émergence des jeux en ligne comme une forme nouvelle et hautement interactive de divertissement; la disponibilité de la musique et de films à la maison via des centres de médias en réseau et plus largement sur Internet via un contenu téléchargeable ou en diffusion; le rôle du contenu généré par l'utilisateur (comme mentionné ci-dessus) comme une nouvelle forme de créativité, par exemple via des services tels que YouTube; la création de nouvelles formes d'art et de divertissement.
Soins de santé	La croissance de l'informatique de la santé en tant que discipline en mettant l'accent sur les dossiers électroniques des patients en ligne et les problèmes de confidentialité connexes; le rôle croissant de la télémédecine dans le soutien du diagnostic à distance, les services avancés tels que la chirurgie à distance (y compris le travail entre équipes de soins); l'application croissante de la technologie des réseaux et des systèmes embarqués dans la vie assistée, par exemple pour surveiller les personnes âgées à leur domicile.
Éducation	L'émergence du e-learning via par exemple des outils web comme les environnements d'apprentissage virtuels; support associé pour apprentissage à distance; soutien à l'apprentissage collaboratif ou communautaire.
Transport et Logistique	L'utilisation de technologies de localisation telles que le GPS dans les systèmes de recherche d'itinéraire et des systèmes de gestion du trafic; la voiture moderne elle-même comme un exemple de système distribué complexe (s'applique également à d'autres moyens de transport tels que les avions); le développement de la carte Web des services tels que MapQuest, Google Maps et Google Earth.
La science	L'émergence du Grid comme technologie fondamentale de l'eScience, y compris l'utilisation de réseaux complexes d'ordinateurs pour prendre en charge le stockage, analyse et traitement de (souvent de très grandes quantités de) données scientifiques; l'utilisation associée du réseau en tant que technologie pour une collaboration mondiale entre groupes de scientifiques.
Gestion de l'environnement	L'utilisation de la technologie des capteurs (en réseau) pour surveiller et gérer l'environnement naturel, par exemple pour fournir une alerte précoce des catastrophes naturelles telles que tremblements de terre, inondations ou tsunamis et coordonner la réponse d'urgence; la collation et l'analyse des paramètres environnementaux pour mieux comprendre les phénomènes naturels complexes tels que le changement climatique.

TABLE 1.1: Domaines d'application sélectionnés et applications en réseau associées [Ca12]

    - Virtuellement : accès à ces fichiers à partir de n'importe quelle machine cliente en faisant « croire » que ces fichiers sont stockés localement
    - Arborescence de fichiers Unix : arborescence unique avec
        - Répertoires physiquement locaux
        - Répertoires distants montés via le protocole NFS (Network File System)[Car08]
- Intérêts
    - Accès aux fichiers à partir de n'importe quelle machine
    - Système de sauvegarde associé à ce serveur
    - Transparent pour l'utilisateur
- Inconvénients: Si le réseau ou le serveur plante : plus d'accès aux fichiers pour personne

Recherche Web

- Un serveur web auquel se connecte un nombre quelconque de navigateurs web (clients)
- Accès à distance à de l'information
    - Accès simple (Serveur renvoie une page HTML statique qu'il stocke localement)
    - Traitement plus complexe (Serveur interroge une base de données pour générer dynamiquement le contenu de la page)
    - Transparent pour l'utilisateur : les informations s'affichent dans son navigateur quelque soit la façon dont le serveur les génère[Car08].

La recherche sur le Web est devenue une industrie de croissance majeure au cours de la dernière décennie, avec des chiffres récents indiquant que le nombre mondial de recherches est passé à plus de 10 milliards par mois du calendrier. La tâche d'un moteur de recherche Web est d'indexer l'ensemble du contenu du World Wide Web, englobant un large éventail de styles d'information, y compris les pages web, sources multimédias et livres (numérisés). C'est une tâche très complexe, car les estimations indiquent que le Web comprend plus de 63 milliards de pages et un billion d'adresses Web uniques. Étant donné que la plupart des moteurs de recherche analysent l'intégralité du contenu Web, puis diffusent un traitement sophistiqué sur cette énorme base de données, cette tâche elle-même représente un défi majeur pour la conception de systèmes distribués.

Google, leader du marché de la technologie de recherche sur le Web, a déployé des efforts considérables pour la conception d'une infrastructure système distribuée sophistiquée pour prendre en charge la recherche (et d'autres applications et services Google comme Google Earth). Cela représente l'une des installations de systèmes distribués les plus grandes et les plus complexes de l'histoire de informatique et exige donc un examen attentif. Points forts de cette infrastructure:

- une infrastructure physique sous-jacente composée d'un très grand nombre d'ordinateurs en réseau situés dans des centres de données du monde entier;
- un système de fichiers distribué conçu pour prendre en charge de très gros fichiers et fortement optimisé pour le style d'utilisation requis par la recherche et d'autres applications Google (en particulier lecture de fichiers à des taux élevés et soutenus);
- un système de stockage distribué structuré associé qui offre un accès rapide à grands ensembles de données;
- un service de verrouillage qui offre des fonctions au système distribuées telles que le verrouillage distribué et accord;
- un modèle de programmation qui prend en charge la gestion de très grands calculs parallèles et distribués sur l'infrastructure physique sous-jacente[Ca12].

Calculs scientifique distribués

- Plusieurs architectures matérielles généralement utilisées
    - Ensemble de machines identiques reliées entre elles par un réseau dédié et très rapide (cluster)
    - Ensemble de machines hétérogènes connectées dans un réseau local ou bien encore par

Internet (grille)
- Principe général
  - Un (ou des) serveur distribue des calculs aux machines clients
  - Un client exécute son calcul puis renvoie le résultat au serveur
- Avantage: Utilisation d'un maximum de ressources de calcul
- Inconvénient: Si le réseau ou le serveur plante, arrête le système [Car08]

1.2.5 Middleware (Intergiciel)

Afin de prendre en charge des ordinateurs et des réseaux hétérogènes tout en offrant un système unique, les systèmes distribués sont souvent organisés au moyen d'une couche de logiciel [Ta07].

Définition 1.2.4 Un intergiciel est la couche logicielle située entre les couches basses (système d'exploitation et protocoles de communication) et la couche des applications dans un système informatique répartis (CORBA, Globus, EJB,, COM, ...) comme indiqué à la Fig.1.1.

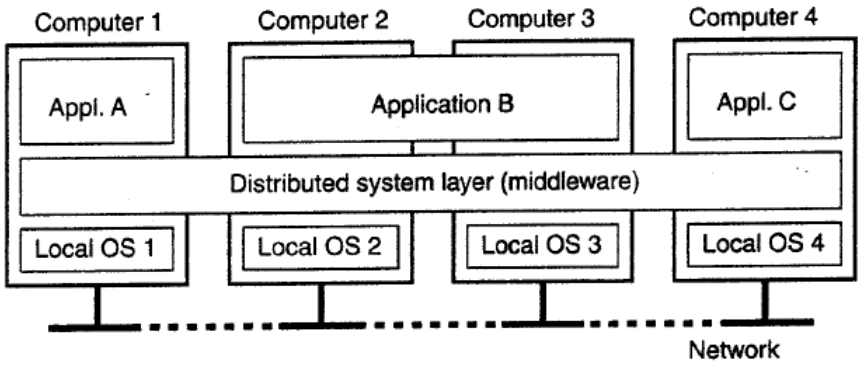


FIGURE 1.1 – Un système distribué organisé en middleware. Le middleware une couche qui s'étend sur plusieurs machines et offre à chaque application la même interface.

La Figure 1.1 montre quatre ordinateurs en réseau et trois applications, dont B est distribué sur les ordinateurs 2 et 3. Chaque application se voit proposer le même interface. Le système distribué fournit les moyens pour les composants d'un unique application distribuée pour communiquer les uns avec les autres, mais aussi pour laisser différentes les applications communiquent. Dans le même temps, il cache, comme meilleur et raisonnable possible, les différences de matériel et de systèmes d'exploitation de chaque application.

1.3 Caractéristiques d'un système distribué

La performance d'un système distribué se révèle dans ces caractéristiques. Ces caractéristiques ci-dessous devraient être prises en compte lors de la conception d'un système distribué [Hut ; MUD].

1.3.1 Interopérabilité

Dans un système distribué, il se pose un vrai problème de coopération entre différents composants du système. En effet, ce problème peut être vu au niveau de la couche matériel (différents réseaux physiques et plateforme matérielle), de la couche système d'exploitation (divers OS utilisés(UNIX, Windows, Mac OS, Solaris)), de la couche application (langages de programmation

différents) et de la couche middleware (.NET pour Microsoft, Corba pour le Consortium OMG). On parle de l'hétérogénéité, un problème dans le partage des ressources dans un système distribué.

L'interopérabilité est une caractéristique importante qui désigne la capacité à rendre compatibles deux systèmes quelconques. A son tour, la compatibilité est la capacité qu'ont deux systèmes à communiquer sans ambiguïté.

En effet, l'interopérabilité vise à réduire le vrai problème de l'hétérogénéité en la masquant par l'utilisation d'un protocole unique de communication (exemple de TCP/IP pour l'Internet). Pour les échanges des messages, il faut utiliser des standards qui cachent les différences entre les différentes plateformes. Actuellement, il existe deux approches principales de standardisation pour masquer l'hétérogénéité : les middlewares et les machines virtuelles. Concrètement, un middleware est représenté par des processus et des ressources d'un ensemble d'ordinateurs, qui interagissent les uns avec les autres. Les middlewares améliorent la communication en offrant des abstractions telles que :

— Le RMI (Remote Method Invocation) qui est la possibilité, pour un objet, d'invoquer la méthode d'un autre objet situé sur une plateforme distante ;
— La notification d'événement pour la propagation d'informations d'une plateforme vers une autre ou plusieurs autres plateformes ou composants d'une application distribuée ;
— La communication entre groupe de processus ;
— La gestion de la réplication des données partagées ;
— La transmission de données multimédia en temps réel.

Les machines virtuelles permettent de supporter le code mobile désignant la possibilité de transfert de code d'une machine source à une machine de destination et son exécution sur cette machine. Au cas des différentes plateformes, le code produit sur l'une ne peut fonctionner sur l'autre. Pour éviter ce problème, le code mobile est généré d'un langage source pour une machine virtuelle donnée (exemple de la JVM). En effet, chaque plateforme doit disposer d'une couche logicielle qui implémente la machine virtuelle car cette dernière représente une généralisation des middlewares offrant les mêmes services.

1.3.2 Partage des ressources

Le partage des ressources est le facteur principal de motivation pour construire les systèmes répartis. Des ressources telles que des imprimantes, des dossiers, des pages Web ou des disques de base de données sont contrôlées par des serveurs du type approprié. Par exemple, les serveurs Web contrôlent des pages Web et d'autres ressources d'enchaînement. Des ressources sont consultées par des clients - par exemple, les clients du web des serveurs s'appellent généralement les browsers (navigateurs) ;

1.3.3 Ouverture

Cette caractéristique fait mention de l'extensibilité dans la mesure où des composants peuvent être ajoutés, remplacés ou supprimés dans un système distribué sans en affecter les autres. Et lorsque nous parlons des composants, nous voyons les matériels (les périphériques, mémoires, interfaces, etc.) et les logiciels (protocoles, pilote, etc.). L'ouverture nécessite que les interfaces logicielles soient documentées et accessibles aux développeurs d'applications. Il se pose un vrai problème avec l'ouverture au sens que les composants d'un système distribué sont hétérogènes. Alors, cette qualité d'ouverture est accordée aux systèmes supportant sans ambages:

1. L'ajout de l'ordinateur au niveau de la couche matérielle;
2. L'ajout de nouveaux services au niveau application, middleware et système d'exploitation;
3. La réimplantation des services anciens.
4. ajout et mise à disponibilité de ressources partagées
5. accès à d'autres applications

6. accès depuis d'autres applications
7. nécessite le libre accès aux spécifications et à la documentation des APIs pour les programmeurs – ex. : RFC pour Internet
   Avantages:
8. développement collaboratif
9. permet une indépendance vis-à-vis des constructeurs et éditeurs de logiciels

1.3.4 Expansibilité

Nous disons qu'un système distribué est expansible lorsque les modifications du système et des applications ne sont pas nécessaires quant à l'augmentation de la taille de ce système.

1.3.5 Performance

Dans ce cas, le système doit s'adapter à bien fonctionner même quand le nombre d'utilisateurs ou de ressources augmentent.

1.3.6 Transparence

« To the user, a distributed system should look exactly like a nondistributed system. » La transparence cache aux utilisateurs l'architecture, la distribution des ressources, le fonctionnement de l'application ou du système distribué pour apparaître comme une application unique cohérente. La norme ISO (1995) définit différents niveaux de transparences telle que la transparence d'accès, de localisation, de concurrence, de réplication, de mobilité, de panne, de performance, d'échelle).

En effet, la transparence n'est pas toujours possible dans certains cas. Notons le cas de la réplication d'une imprimante des caractéristiques différentes pour besoin des performances dans le système. Cependant, l'utilisateur doit toutefois avoir la possibilité de spécifier concrètement sur quelle imprimante il souhaite imprimer ses documents.

1.3.7 Sécurité

Le problème de sécurité se pose dans tout système informatique. Dans un système distribué, les ressources doivent être protégées contre des utilisations abusives et malveillantes. En particulier, le problème de piratage des données sur le réseau de communication. En ces raisons, il est préférable d'utiliser des périphériques ou logiciels licenciés. Outre, les connexions doivent être sécurisées par authentification avec les éléments distants ainsi que les messages circulant sur ce réseau doivent être cryptés en vue d'éviter des conséquences graves. Le concept de sécurité des systèmes d'information recouvre un ensemble de méthodes, techniques et outils chargés de protéger les ressources d'un système d'information afin d'assurer:

- la disponibilité des services : les services (ordinateurs, réseaux, périphériques, applications...) et les informations (données, fichiers...) doivent être accessibles aux personnes autorisées quand elles en ont besoin ;
- la confidentialité des informations : les informations n'appartiennent pas à tout le monde ; seuls peuvent y accéder ceux qui en ont le droit ;
  - protection contre les destinataires indésirables Intégrité
  - protection contre l'altération ou la corruption des données Authentification, signature électronique
  - identification des partenaires
  - non-déni d'envoi ou de réception
  - messages authentifiés
  - respect possible de l'anonymat
- l'intégrité des systèmes : les services et les informations (fichiers, messages...) ne peuvent être modifiés que par les personnes autorisées (administrateurs, propriétaires...).

Type de transparence	Signification
Transparence d'accès	il s'agit d'utiliser les mêmes opérations pour l'accès aux ressources distantes que pour les ressources locales ; — les ressources locales et distantes doivent pouvoir être accessibles de la même manière. — ex. : système de montage de volumes Unix
Transparence à la localisation	l'accès aux ressources indépendamment de leur emplacement doit être inconnue à l'utilisateur ; — L'utilisateur ne connaît pas où sont situées les ressources — les ressources doivent être accessibles quelle que soit leur localisation physique — ex. : URL Web
Transparence à la concurrence	il s'agit de cacher à l'utilisateur l'exécution possible de plusieurs processus en parallèle avec l'utilisation des ressources partagées en évitant des interférences;
Transparence à la réplication	la possibilité de dupliquer certains éléments/ressources (fichiers de base de données) pour augmenter la fiabilité et améliorer les performances doit être cachée à l'utilisateur ; — plusieurs instances des ressources doivent être déployées pour assurer la fiabilité du système. — L'utilisateur ne connaît pas le nombre de copies existantes
Transparence de mobilité	il s'agit de permettre la migration des ressources et des clients à l'intérieur d'un système sans influencer le déroulement des applications ; les ressources peuvent être déplacées sans modification de leur nom.
Transparence de panne	il s'agit de permettre aux applications des utilisateurs d'achever leurs exécutions malgré les pannes qui peuvent affecter les composants d'un système (composants physiques ou logiques) ; une panne ne doit pas bloquer le fonctionnement global du système
Transparence à la modification de l'échelle (scalabilité)	il s'agit de la possibilité d'une extension importante d'un système sans influence notable sur les performances des applications; le système et les applications doivent pouvoir supporter les changements d'échelles sans modification interne des algorithmes par exemple; — Capacité à rester efficace en cas de forte augmentation des ressources et des utilisateurs — Adapter le dimensionnement de l'application — ajout/réplication de composants
Transparence à la reconfiguration	il s'agit de cacher à l'utilisateur la possibilité de reconfigurer le système pour en augmenter les performances en fonction de la charge ;

TABLE 1.2: Les différents niveaux de transparences

— Capacité à assurer la sécurité des données qui transitent, l'authentification des participants et à empêcher les intrusions

1.3.8 Concurrence

Le problème de la concurrence permet l'accès simultané à des ressources matérielle ou logicielle par plusieurs processus: ex. : base de données, serveur Web, etc. Ce problème se pose pour les systèmes distribués comme pour les systèmes centralisés. En effet, il y a bien d'autres ressources dont l'accès simultané n'est pas possible. Dans ce cas, leur manipulation ne peut se faire que par un processus à la fois. Le cas des ressources physiques telles que l'imprimante mais aussi des ressources logiques telles que les fichiers, les tables des bases de données, etc. Dans ce cas, les applications distribuées (reparties)actuelles autorisent l'exécution de plusieurs services en concurrence (cas de l'accès à une base de données). Chaque demande est prise en compte par un processus simple appelé thread ; et la gestion de la concurrence fait appel aux mécanismes de synchronisation classiques.

Approches possibles:

redondance
– ex. : duplication d'une base de données
– problème de synchronisation entre les ressources

gestion des ressources par le système d'exploitation
– ex. : accès à un fichier partagé
– problème de synchronisation entre processus, utilisation de sémaphores, mécanismes de synchronisation de threads
– synchronisation trop forte = risque de « sérialisation »

1.3.9 Tolérance aux pannes

Une panne peut être comprise comme une faille au sein du système pouvant conduire à des résultats erronés comme aussi engendrer l'arrêt de toute ou partie d'un système distribué. Les pannes peuvent résulter des différentes couches et se propager éventuellement aux autres. Peut-être, c'est une raison matérielle ou logique liée à la conception des applications, des middlewares et des systèmes d'exploitation. Ainsi, un système distribué doit être conçu pour masquer ce genre des pannes aux utilisateurs. La panne de certains serveurs (ou leur réintégration dans le système après la réparation) ne doit pas perturber l'utilisation du système en terme de fonctionnalité.

La tolérance aux pannes est la capacité de l'application à s'exécuter en environnement dégradé.
Les pannes généralement partielles et d'origines diverses
— panne d'un composant
— panne d'un lien de communication entre composants
Limiter les conséquences liées à la panne d'un système matériel (ou logiciel)
— éviter une trop forte centralisation
— éviter d'isoler des fonctions vitales

Indicateur de performance = disponibilité: proportion de temps pendant laquelle il est disponible pour utilisation mesurée en pourcentage disponibilité de 99,999% pour les systèmes les plus fiables

Tolérance aux pannes – approches possibles

la détection de pannes: but = détecter que les données reçues ne sont pas celles attendues
ex. : bit de contrôle pour l'envoi de données

le masquage des pannes: but = limiter l'impact des pannes
ex. : retransmission de messages, systèmes de cache

la tolérance aux pannes: alerter si panne trop importante pour être corrigée
ex. : serveur Web en panne

la reprise sur erreur: rétablir les données suite à une panne
ex. : systèmes de sauvegardes des données permanentes

la redondance: duplication de composants pour en avoir toujours au moins un de disponible
ex. : plusieurs chemins entre routeurs, plusieurs BDs

1.3.10 Disponibilité

Dans un système distribué, l'indisponibilité d'un seul composant du système (serveur, base de données, ...) peut rendre indisponible le système complet alors qu'il doit rendre en permanence des services et d'une façon correcte. On mesure alors la disponibilité de ce type de système à celle de son maillon le plus faible. Ces risques d'indisponibilité du système peuvent être dus :

- aux pannes empêchant le système ou à ses composants de fonctionner correctement ;
- aux surcharges dues à des sollicitations excessives d'une ressource ;
- aux attaques de sécurité pouvant causer, d'une façon ou d'une autre, des dysfonctionnements, les incohérences et pertes de données et même l'arrêt du système.

Pour couvrir ce risque, plusieurs solutions peuvent être envisageables:

- mettre en place en amont une architecture permettant d'assurer la disponibilité cible pour tous les composants. Une fois que cette architecture est en production, des opérateurs doivent à l'aide de logiciels s'assurer de la détection au plus tôt d'une défaillance de l'un des composants de l'architecture.
- veiller à la réplication des données au sein de ce système

1.3.11 Hétérogénéité

- au niveau des réseaux: réseaux de différents types
- au niveau du matériel informatique: codage des données différent suivants les architectures
- au niveau des systèmes d'exploitation: protocoles standards mais APIs différentes d'un OS à l'autre
- au niveau des langages de programmation: différences pour le codage des caractères ou les structures de données
- au niveau des implantations: nécessité de suivre les standards établis

Hétérogénéité – approches possibles

- protocoles standards de communication masquent l'hétérogénéite des systèmes
  ex. : standards pour l'Internet TCP/IP
- utilisation de middlewares: modèles de programmation distribuée standardisés et uniformes accès aux BDs, appel d'objets à distance implémentés au-dessus des standards de l'Internet
  ex. : CORBA, Java RMI, Microsoft COM/DCOM, etc.
- utilisation de code mobile envoyé d'un ordinateur à l'autre exécuté sur l'ordinateur distant
  ex. : applets Java, agents mobiles

1.4 Comparaison (Centralisé, Distribué)

Les deux concepts ne sont cependant pas opposés
Modèles mixtes :

Hiérarchique (ex : un maître délègue aux serveurs secondaires)

Redondance (ex : serveur de backup)

Super-nodes Le choix dépend des difficultés de réalisation : pragmatisme [PHI]

Centralisé	**Distribué**
Système centralisé : tout est localisé sur la même machine et accessible par le programme (Serveur centralisé avec terminaux) — Système logiciel s'exécutant sur une seule machine accédant localement aux ressources nécessaires (données, code, périphériques, mémoire...) — Serveur coûteux — Engorgement du serveur — Contrôle centralisé — Horloge commune, mémoire commune — Sécurité	Système distribué : Ensemble d'ordinateurs indépendants connectés en réseau et communiquant via ce réseau, cet ensemble apparaît du point de vue de l'utilisateur comme une unique entité — Faible coût d'achat — Autonomie mais manque de service de coordination — Contrôle décentralisé — Pas d'horloge commune, pas de mémoire commune — Pas d'état global — Exploitation du parallélisme — Mécanisme de détection de défauts et de reprise — Vulnérabilité
Caractérisation: Ensemble de composants (coeurs, processeurs, périphériques) partageant une mémoire et une horloge. — Possibilité de dater de manière non ambiguë les évènements — Possibilité d'accéder à une donnée commune — Partage de ressources : processeur, mémoire, disque, périph, ... — Programmation concurrente synchronisée : sémaphore, mutex. Exemple : problème de synchronisation en Java multithread	**Caractérisation:** Ensemble de composants (ordinateurs,téléphones,embarqués,...) autonomes reliés par un réseau de communication, sans partage. — Programmation distribuée — Pas d'état global : un processus ne peut pas connaître instantanément l'état des autres processus — Pas d'horloge globale : chaque processus a sa propre horloge — Échanges : communication et synchronisation, uniquement par échange de messages — Risque de perte de messages et Risque de fautes
Centralisé souvent plus facile mais... — Concentration en un point conduit à la surcharge — Fragilité face à la tolérance aux pannes — Pas toujours adapté aux besoins (ex : traitements locaux)	**Avantages:** — Accès et partage de ressources distantes (ressources physiques: imprimantes, espace disque, processeurs, etc), (ressources logiques: fichiers, données textuelles, audio, images,vidéo) — La distribution permet la mise à disposition de services — Répartition géographique: ex : système de réservation avec des centres dans différents lieux — Puissance de calcul: connexion de machines en réseau permet d'obtenir une puissance de calcul à moindre coût — Disponibilité: relative indépendance des défaillances des différents sites — Flexibilité: architecture distribuée plus modulaire **Inconvénients** — Pas d'horloge globale, pas d'état global immédiat accessible à un site — Fiabilité relative: distribution permet d'introduire de la tolérance aux fautes — Sécurité relative: architecture distribuée plus difficile à protéger (plusieurs points d'accès aux ressources, évolution dynamique de l'architecture) — Difficulté de mise en œuvre — N'évite pas tous les risques de surcharge (ex : diffusion systématique de messages)

1.5 Résumé du chapitre

Les systèmes distribués sont constitués d'ordinateurs autonomes qui fonctionnent ensemble pour donner l'apparence d'un seul système cohérent. Un avantage important est qu'ils facilitent l'intégration de différentes applications s'exécutant sur différents ordinateurs en un seul système. Un autre avantage est que, lorsqu'il est correctement conçu, le système distribué s'adapte bien à la taille du réseau sous-jacent. Ces avantages se font souvent au prix de logiciels plus complexes, de dégradations de performance, et aussi souvent une sécurité plus faible.

Les systèmes distribués visent souvent à masquer un bon nombre des subtilités liées à la distribution des processus, des données et du contrôle.

Il existe différents types de systèmes distribués qui peuvent être classés comme étant orienté vers les calculs et le traitement de l'information.

1.6 Exercices

Exercice 1.1 Quelles sont les principales différences entre un système parallèle et un système distribué? ■

Exercice 1.2 Identifier quelques applications distribuées dans le domaine scientifique et commercial. Pour chaque application, déterminez lequel des facteurs de motivation sont importants pour construire l'application sur un système distribué. ■

Exercice 1.3 Donnez cinq types de ressources matérielles et cinq types de données ou de ressources logicielles qui peuvent utilement être partagé. Donnez des exemples de leur partage tel qu'il se produit dans la pratique dans les systèmes distribués. ■

Exercice 1.4

1. Quel est le rôle du middleware dans un système distribué ?
2. Expliquez ce que l'on entend par transparence (de distribution) et donnez des exemples des différents types de transparence.
3. Pourquoi n'est-il pas toujours judicieux de viser à mettre en œuvre le plus haut degré de transparence possible?

■

Exercice 1.5

1. Qu'est-ce qu'un système distribué ouvert et quels sont les avantages de l'ouverture ?
2. Décrivez précisément ce que l'on entend par système évolutif.
3. L'évolutivité peut être obtenue en appliquant différentes techniques. Quelles sont ces techniques ?

■

2. Modèles de systèmes distribués

Ce chapitre traite des caractéristiques des protocoles de communication entre processus dans un système distribué, c'est-à-dire de communication interprocessus.

Les primitives de communication interprocessus sont décrites prennent toutes en charge point à point communication, mais il est également utile de pouvoir envoyer un message depuis un expéditeur à un groupe de récepteurs. Le chapitre considère également la communication multicast, y compris Multidiffusion IP et les concepts clés de fiabilité et d'ordre des messages en multidiffusion la communication.

2.1 Processus communicant (définition, caractéristique, chronogrammes)

2.1.1 Notion d'asynchrone et de synchrone par référence à une horloge

Un système est synchrone si les évènements qu'il traite sont rythmés selon une horloge. Un système asynchrone traite les évènements aléatoires selon leur instant de présentation.

Dans un système synchrone, les opérations (instructions, calculs, logique, etc.) sont coordonnées par un ou plusieurs signaux d'horloge centralisés. Un système numérique asynchrone, en revanche, n'a pas d'horloge globale. Les systèmes asynchrones ne dépendent pas d'heures d'arrivée strictes des signaux ou des messages pour un fonctionnement fiable. La coordination est obtenue via des événements tels que: arrivée de paquets, changements (transitions) de signaux, protocoles de négociation et autres méthodes.

Les systèmes distribués ont une évolution asynchrone (pas de mémoire commune, pas d'horloge commune)

L'état d'un site ne peut être connu que par les informations véhiculés par les messages

Chaque site à sa propre horloge, la notion d'état global n'existe pas.

2.1.2 Système synchrone / asynchrone

Un modèle synchrone est un modèle où

— les contraintes temporelles sont bornées

— On sait qu'un processus évoluera dans un temps borné

— On sait qu'un message arrivera en un certain délai
— On connaît la limite de dérive des horloges locales
Un modèle asynchrone n'offre aucune borne temporelle
— Modèle bien plus contraignant et rendant impossible ou difficile la réalisation de certains algorithmes distribués
— Exemple : ne sait pas différencier en asynchrone
  — Le fait qu'un processus est lent ou est planté
  — Du fait qu'un message est long à transiter ou est perdu

2.1.3 Exécutions synchrones et asynchrones

En plus des deux classifications de synchronisation/asynchronie du processeur et des primitives de communication synchrone/asynchrone, il existe une autre classification, à savoir celle des exécutions synchrones / asynchrones.

— Une exécution asynchrone est une exécution dans laquelle (i) il n'y a pas de processeur synchronie et il n'y a pas de limite sur la vitesse de dérive du processeur horloges, (ii) les délais de message (temps de transmission + temps de propagation) sont finis mais illimité, et (iii) il n'y a pas de limite supérieure sur le temps pris par un processus pour exécuter une étape.
— Une exécution synchrone est une exécution dans laquelle (i) les processeurs sont synchronisés et taux de dérive d'horloge entre deux processeurs quelconques est bornée, (ii) Délivrance des messages (transmission + délivrance) sont telles qu'elles se produisent en une étape ou un tour logique, et (iii) il existe une limite supérieure connue sur le temps mis par un processus pour exécuter une étape. [Aja08]

2.1.4 Modèle conceptuel d'un système distribué

Les éléments de base d'un système distribués sont:
— Processus communiquant
— Canaux de communication

Définition 2.1.1 Site logique ou processus: Élément logiciel effectuant une tâche donnée ou un calcul,
— Exécution d'un ensemble d'instructions
— Une instruction correspond à un événement local au processus
— Dont les événements d'émission et de réception de messages
— Les instructions sont généralement considérées comme atomiques
possède:
— Un identifiant unique
— Un état local
— Une mémoire locale
— Une horloge locale
— Ensemble de ses données et des valeurs de ses variables locales
Connaît par hypothèse
— Les processus avec qui il peut communiquer et leur nombre (voisins)
— Les procédures de communication
— Éventuellement l'état approximatif des autres processus
— **tâche** : ensemble d'instructions
— **instruction** : correspond à un évènement local du processus
— **évènement local** : changement d'état du processeur, émission de message et réception de message.

[PHI]

Définition 2.1.2 Canal de communication: Canal logique de communication point à point pour communication entre 2 processus et pour le transit de messages sur le canal

Caractéristiques d'un canal

- Uni ou bi-directionnel
- Fiable ou non : perd/modifie ou pas des messages
- Ordre de réception par rapport à l'émission
- Ex. : FIFO = les messages sont reçus dans l'ordre où ils sont émis
- Synchrone ou asynchrone
  1. *Synchrone : l'émetteur et le récepteur se synchronisent pour réaliser l'émission et/ou la réception
  2. * Asynchrone : pas de synchronisation entre émetteur et récepteur
- Taille des tampons de message cotés émetteur et récepteur
- Limitée ou illimitée
- Modèle généralement utilisé: Fiable, FIFO, tampon de taille illimitée, asynchrone (en émission et réception) et bidirectionnel
- Variante courante avec réception synchrone **Exemple : modèle des sockets TCP**
  - Fiable
  - FIFO
  - Bidirectionnel
  - Synchrone pour la réception
  - On reçoit quand l'émetteur émet
  - Sauf si données non lues dans le tampon coté récepteur
  - Asynchrone en émission
  - Emetteur n'est pas bloqué quand il émet quoique fasse le récepteur [Car08].

Problème : communication inter-processus

- dans un système centralisé : communiquer par la mémoire partagée
- dans un système distribué : en général, l'absence de la mémoire partagée, on doit communiquer par messages avec des protocoles

  Protocole: ensemble de règles qui gère les communication inter-processus dans un système distribué
  - il détermine un l'accord sur la façon dont les communications doivent d'effectuer
  - les protocoles de communication (modèle OSI)
  - le modèle Client / Serveur
  - les Appels de procédure à distance (RPC=Remote Procedure Call)

Définition 2.1.3 Chronogramme: un chronogramme décrit l'ordonnancement temporel des évènements des processus et des échanges de messages

- Chaque processus est représenté par une ligne
- Trois types d'évènements signalés par une ligne
  1. Émission d'un message à destination d'un autre processus
  2. Réception d'un message venant d'un autre processus
  3. Évènement interne dans l'évolution d'un processus, voir figure 2.1

Règle de numérotation d'un évènement: e_{xy}: avec x le numéro du processus et y le numéro de l'évènement pour le processus dans l'ordre croissant, voir figure 2.2.

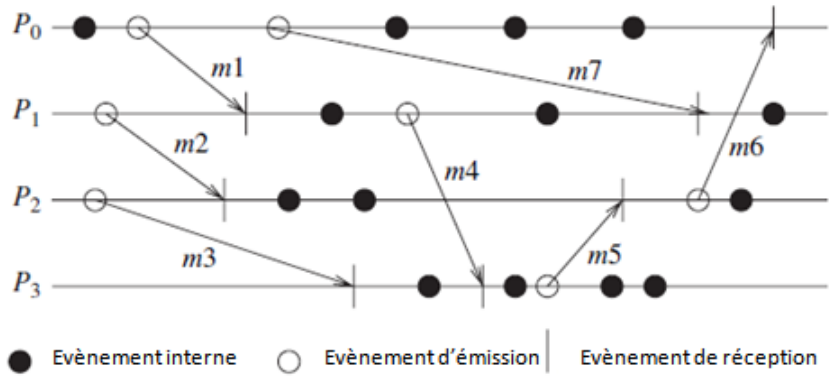


FIGURE 2.1 – Chronogramme des évènements

2.2 Notion de causalité (d'événements, des messages, délivrance FIFO)

2.2.1 Dépendance causale

Définition 2.2.1 Relation de dépendance causale il y a une dépendance causale entre deux événements si un événement doit avoir lieu avant l'autre

Notation : $e \rightarrow e'$; e doit se dérouler avant e'

Si $e \rightarrow e'$, alors une des trois conditions suivantes doit être vérifiée pour e et e'

— Si e et e' sont des événements d'un même processus, e précède localement e'

— Si e est l'émission d'un message, e' est la réception de ce message

— Il existe un événement f tel que $e \rightarrow f'$ et $f \rightarrow e'$ [Car08]

Les dépendances causales définissent un **des ordres partiels** pour des ensembles d'évènements du système.

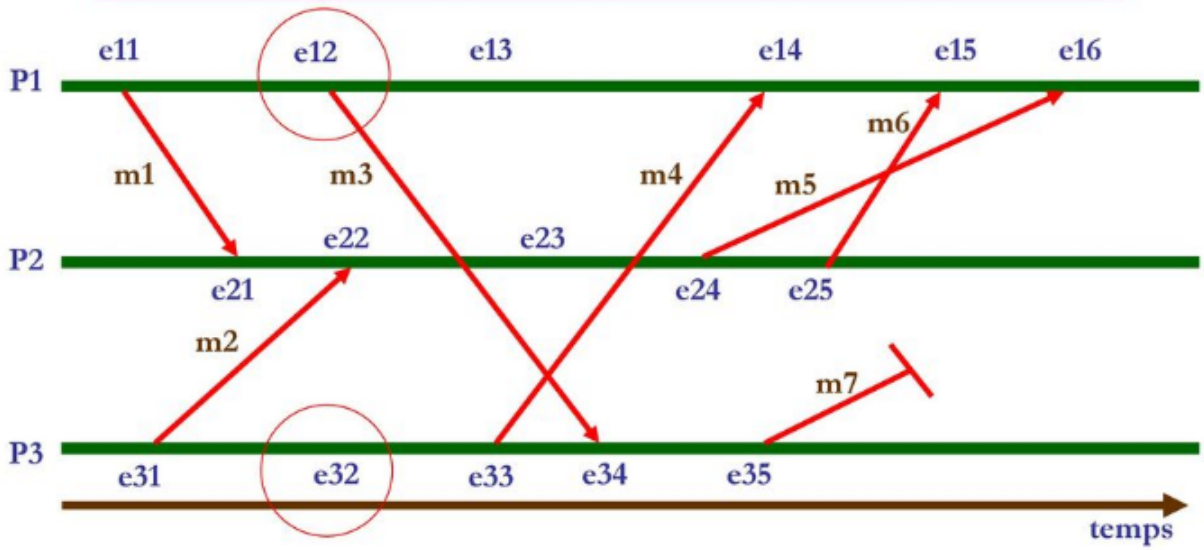


FIGURE 2.2 – Trois processus tous reliés entre-eux par des canaux, temps de propagation des messages quelconques et possibilité de perte de message.

Sur l'exemple de la figure 2.2 : Quelques dépendances causales autour de e12

— Localement : $e11 \rightarrow e12$, $e12 \rightarrow e13$
— Sur message : $e12 \rightarrow e34$
— Par transitivité : $e12 \rightarrow e35$ (car $e12 \rightarrow e34$ et $e34 \rightarrow e35$).
— Dépendance causale entre $e12$ et $e32$?
A priori non : absence de dépendance causale,
Des événements non liés causalement se déroulent en parallèle

Définition 2.2.2 Relation de parallélisme : ||
— $e||e' \Leftrightarrow \neg((e \rightarrow e') \vee (e' \rightarrow e))$
Parallélisme logique : ne signifie pas que les 2 événements se déroulent simultanément mais qu'il peuvent se dérouler dans n'importe quel ordre.

2.2.2 Notion de précédence causale

Cette relation définit un ordre partiel des évènements
— Des évènements e et e' non comparables sont dit concurrents, ce qu'on note par $e||e'$
— A un évènement e on peut alors associer trois ensembles d'évènements:
Passé(e): ensemble des évènements antérieurs à e dans l'ordre causale (e appartient à cet ensemble)
Futur(e): ensemble des évènements postérieurs à e dans l'ordre causale (e appartient à cet ensemble)
Concurrent(e): ensemble des évènements concurrents avec e.

■ **Exemple 2.1** En se référant à la figure 2.2
— Passé $(e_{32}) = \{e_{32}, e_{31}\}$
— Futur$(e_{32}) = \{e_{32}, e_{33}, e_{34}, e_{35}, e_{14}, e_{15}, e_{16}\}$
— concurrents$(e_{32}) = \{e_{11}, e_{12}, e_{13}, e_{21}, e_{22}, e_{23}, e_{24}, e_{25}\}$

■

2.2.3 Propriété de causalité élémentaire

Par la nature physique de la communication, l'émission d'un message sur un site précède nécessairement la réception du message sur le site destinataire.

Toute réception d'un message est causée par une émission antérieure. (il ne peut y avoir de réception spontanée de message).

Cette relation causale permet d'établir, dans un système réparti, une relation d'ordre partiel entre l'événement d'émission d'un message sur un site et l'événement de réception du message sur un autre site destinataire. cette relation se note $(\rightarrow)$ (on lit : a précédé) :

$$m, \acute{E}MISSION(m) \rightarrow R\acute{E}CEPTION(m) \tag{2.1}$$

Plus exactement la relation est établie lorsque le message a été reçu

$$m, R\acute{E}CEPTION(m) \Rightarrow (\acute{E}MISSION(m) \rightarrow R\acute{E}CEPTION(m)) \tag{2.2}$$

2.2.4 Dépendance causale entre messages

Délivrance FIFO

Cette propriété assure que si deux messages sont envoyé successivement de P_i vers un même destinataire P_j, le premier sera délivré à P_j avant le second (voir figure 2.3) .

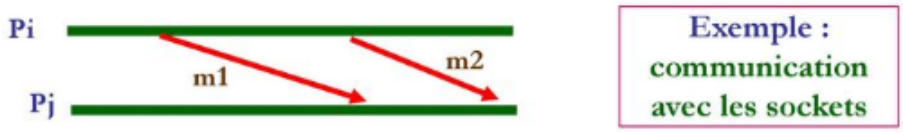


FIGURE 2.3 – Délivrance FIFO.

Délivrance causale

Cette propriété étend la précédente à des communication à destination d'un même processus en provenance de plusieurs autre.

Elle assure que si l'envoi du message m_1 par P_i à destination de P_k précède (causalement) l'envoi du message m_2 par P_j à destination de P_k, le message m_1 sera délivré avant le message m_2 à p_k (voir figure 2.4) .

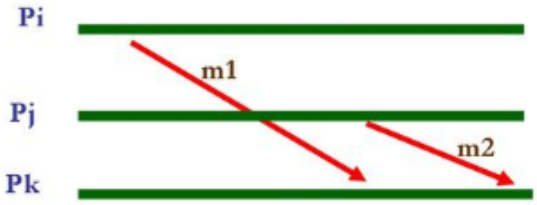


FIGURE 2.4 – Délivrance causale.

DÉPENDANCE CAUSALE ENTRE MESSAGES

Les messages respectent la dépendance causale si et seulement si :
$\forall P_i, \forall P_j, \forall P_k, \exists m$ *émis sur* C_{ij}, m_1 *émis sur* C_{kj},

$$ÉMISSION_i(m) \rightarrow ÉMISSION_k(m_1) \Rightarrow RÉCEPTION_j(m) \rightarrow RÉCEPTION_j(m_1)$$

C_{ij}: Canal de communication entre le site i et le site j
P_i: processus i
m, m_1: messages

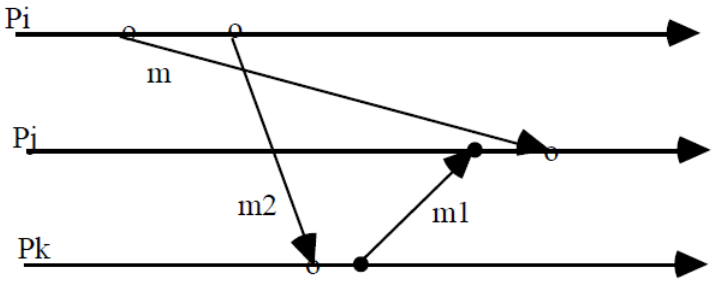


FIGURE 2.5 – Trois processus tous reliés entre-eux par des canaux, temps de propagation des messages quelconques et possibilité de perte de message.

■ **Exemple 2.2** la réception sur P_j ne respecte pas la dépendance causale ■

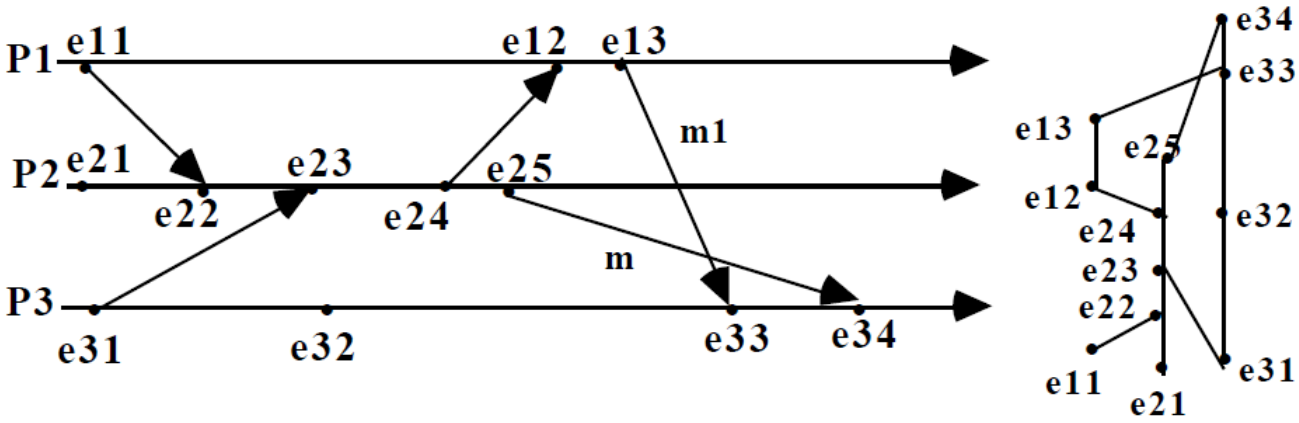


FIGURE 2.6 – Trois processus tous reliés entre-eux par des canaux, temps de propagation des messages quelconques et possibilité de perte de message.

■ **Exemple 2.3** la réception respecte la dépendance causale car les émissions e13 et e25 ne sont pas en relation de précédence [Kai]. ■

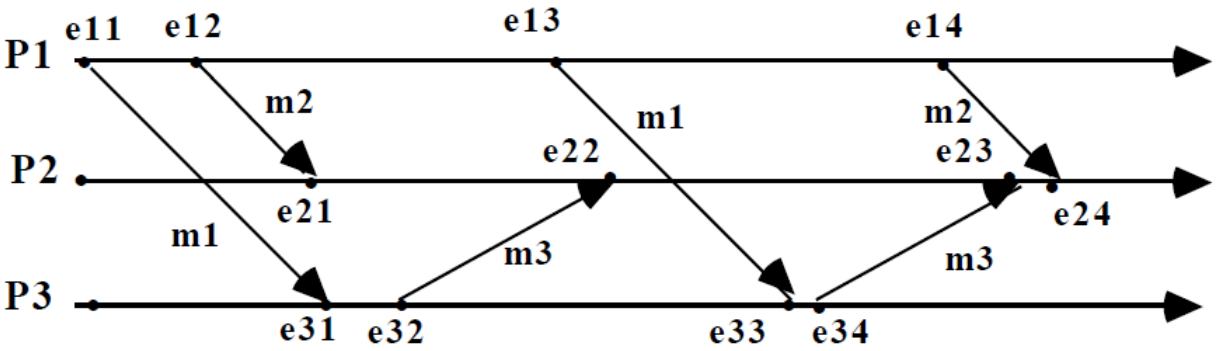


FIGURE 2.7 – Trois processus tous reliés entre-eux par des canaux, temps de propagation des messages quelconques et possibilité de perte de message.

■ **Exemple 2.4** pas de dépendance causale entre

$$\acute{E}MISSION_1(m_2) \rightarrow \acute{E}MISSION_3(m_3)$$

Donc il n'y a pas de dépendance causale entre m_2 et m_3. ■

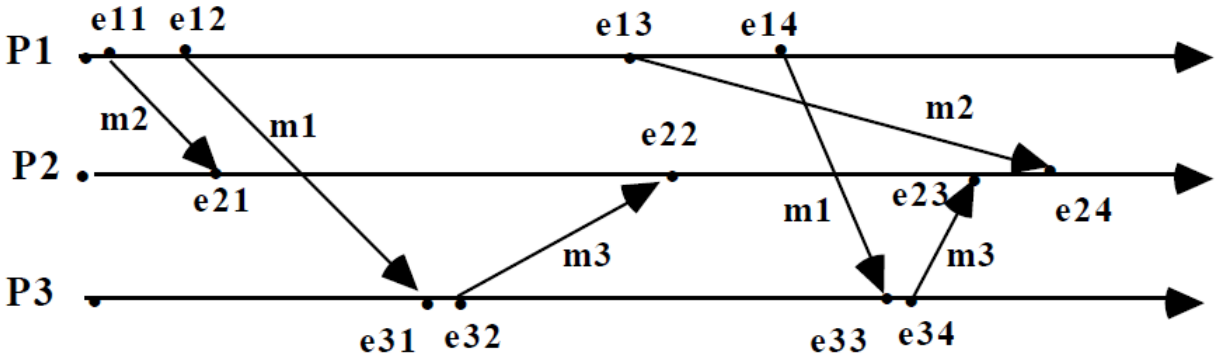


FIGURE 2.8 – Trois processus tous reliés entre-eux par des canaux, temps de propagation des messages quelconques et possibilité de perte de message.

■ **Exemple 2.5**

$$\acute{E}MISSION_1(m_2) \rightarrow \acute{E}MISSION_3(m_3) \Rightarrow R\acute{E}CEPTION_2(m_2) \rightarrow R\acute{E}CEPTION_2(m_3)$$

Donc il y a dépendance causale entre m_2 et m_3 ■

2.3 Communication inter-processus

2.3.1 La communication synchrone ou asynchrone

Le dictionnaire « LEXIS » donne la définition suivante de l'adjectif asynchrone employé dans le contexte de l'informatique : « Se dit d'un ordinateur dans lequel chaque opération est initialisée par un signal provoqué soit par l'achèvement de l'opération précédente, soit par la libération des organes nécessaires. » Mais cette définition ne suffit pas à différencier les communications synchrones des communications asynchrones, puisque cette définition s'applique aussi dans le cas d'un système synchrone. Toutefois, le système synchrone est rythmé par une horloge unique alors que le système asynchrone ne privilégie pas l'horloge pour cadencer l'information. Les notions de synchrone et d'asynchrone sont très délicates à manier. On peut voir deux niveaux de définition qui se rejoignent selon que l'on utilise la notion d'horloge ou la notion de borne pour caractériser les données temporelles.

Notion d'asynchrone et de synchrone par référence à des bornes temporelles

Un système est synchrone si toutes les grandeurs temporelles le caractérisant peuvent être bornées. Les délais de transmission des messages sont contingentés, la vitesse des processus est bridée, la dérive des horloges est supervisée. Un système asynchrone traite les événements alors qu'il s'avère impossible de fixer des bornes sur certaines caractéristiques temporelles. Par exemple, une boîte aux lettres est ouverte à un instant qui reste à la discrétion totale de son propriétaire.

Dans le domaine des systèmes répartis, l'envoi et la réception des messages sont liés de manière inséparable au mode asynchrone qui est le mode naturel de communication entre ressources distantes. Cependant, c'est le point de vue métier qui perçoit le système comme un ensemble de ressources distantes. Par ailleurs, les échanges de messages sont réalisés au moyen de signaux et des deux (2) primitives principales « envoyer » et « recevoir ».

Différentes techniques de communication existent et correspondent à des besoins différents qui varient selon les protocoles.

Définition 2.3.1 Un protocole: est un ensemble de règles et d'étapes successives suivies par chaque entité reliée au canal de communication.

2.3.2 Typologie de communication

Classons les différentes techniques de communication par type et détaillons les successivement.

Diffusion sélective (multicast)

Dans le cas de la communication par diffusion sélective de type multicast, le système est composé d'un ensemble de processus qui sont séparés dans l'espace et qui communiquent au moyen d'un réseau. Ainsi un processus fournit explicitement la liste des destinataires et envoie le message simultanément à tous les processus concernés et à aucun autre. Dans l'illustration suivante le processus A envoie le message simultanément au processus B et au processus D. Il n'envoie pas le message au processus C. [Tan95].

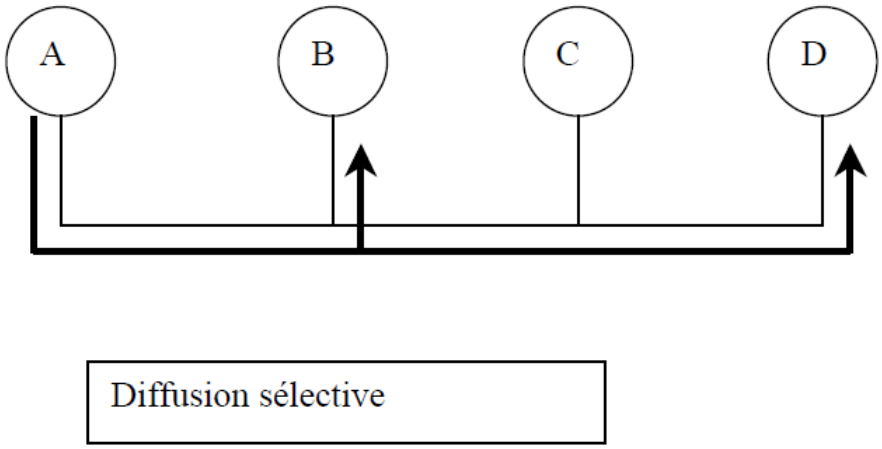


FIGURE 2.9 – Diffusion sélective (multicast).

Diffusion générale (broadcast)

Dans le cas de la communication par diffusion générale de type broadcast, un processus envoie le message simultanément à tous les autres processus. Même ceux qui ne sont pas concernés reçoivent le message et doivent alors détruire ce message dont ils n'ont pas besoin. Dans une approche de communication par broadcast, la sécurité est plus difficile à implanter que dans une approche de communication à diffusion sélective. Dans l'illustration suivante le processus A envoie le message simultanément au processus B, au processus C et au processus D. Un dispositif spécial affecte le processus C au niveau du micro-noyau. Si le processus C ne fait pas partie du groupe, le message est détruit [Tan95].

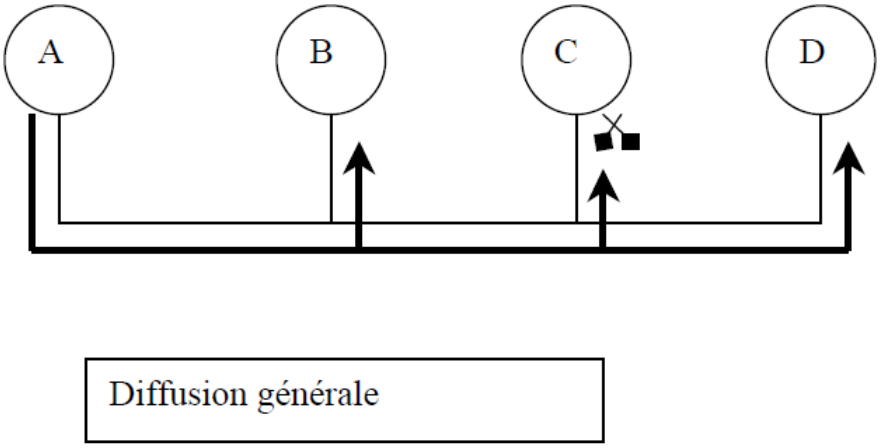


FIGURE 2.10 – Diffusion générale (broadcast).

Communication point à point (unicast)

Dans le cas de la communication en point à point de type unicast, un processus connait la liste des destinataires et envoie le message explicitement à chacun des processus distants concernés et à aucun autre [Tan95].

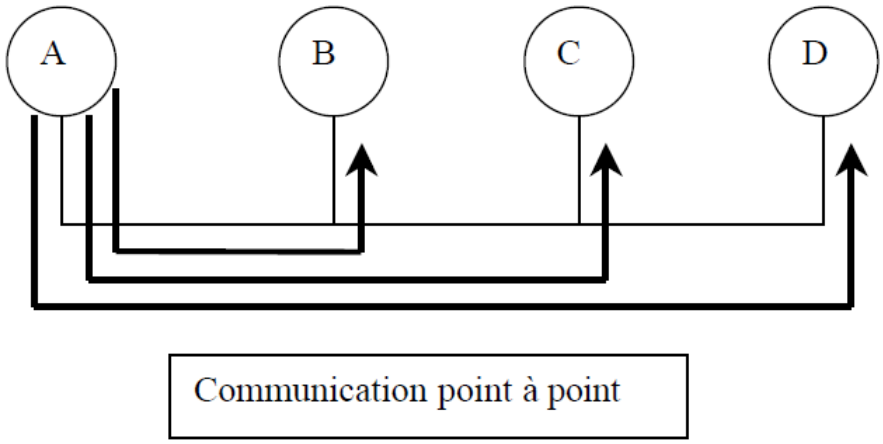


FIGURE 2.11 – Communication point à point.

2.4 Conclusion

Les sous-sections précédentes ont identifié les grands principes sous-jacents à toute les primitives de communication. Dans cette section, nous mentionnons brièvement certains interfaces disponibles qui incarnent certains des concepts ci-dessus.

Il existe une large gamme de primitives pour la transmission de messages. De nombreux produits logiciels commerciaux(banque, paie, etc., applications) utilisent des bibliothèques primitives fournies avec le logiciel commercialisé par les fournisseurs (par exemple, le Logiciel IBM CICS qui dispose d'une base de clients très largement installée dans le monde utilise ses propres primitives). La bibliothèque MPI (Message-Passing Interface) et la bibliothèque PVM (machine virtuelle parallèle) sont largement utilisées par la communauté des scientifiques, mais d'autres bibliothèques alternatives existent. Logiciel commercial est souvent écrit en utilisant le mécanisme des appels de procédure distante (RPC) dans lequel les procédures qui résident potentiellement sur le réseau sont

invoquées de manière transparente à l'utilisateur, de la même manière qu'une procédure locale est invoquée.

Sous les couvertures, les primitives de socket ou les primitives de couche de transport de type socket sont invoqué pour appeler la procédure à distance. Il existe de nombreuses implémentations de RPC, - par exemple, Sun RPC et environnement informatique distribué (DCE) RPC. «Messagerie» et «streaming» sont deux autres mécanismes de la communication. Avec la croissance des logiciels basés sur les objets, des bibliothèques pour invocation de méthode (RMI) et invocation d'objet distant (ROI) avec leurs propres un ensemble de primitives est proposé et normalisé par différentes agences. CORBA (architecture de courtier de demande d'objet commun) et DCOM (distribué modèle objet composant) sont deux autres architectures standardisées avec leur propre ensemble de primitives. De plus, plusieurs projets dans la recherche sont en train de concevoir leur propre saveur de primitives de communication.

2.5 Exercices

Exercice 2.1 On considère un système distribué 2.12 constitué de 4 sites P_1, P_2, P_3, P_4, s'envoyant des messages de façon asynchrone comme représenté par la figure suivante. Les événements d'un processus, représentés par des gros points noirs, sont soit des événements locaux (étapes d'un calcul), soit des envois ou des réceptions de messages.

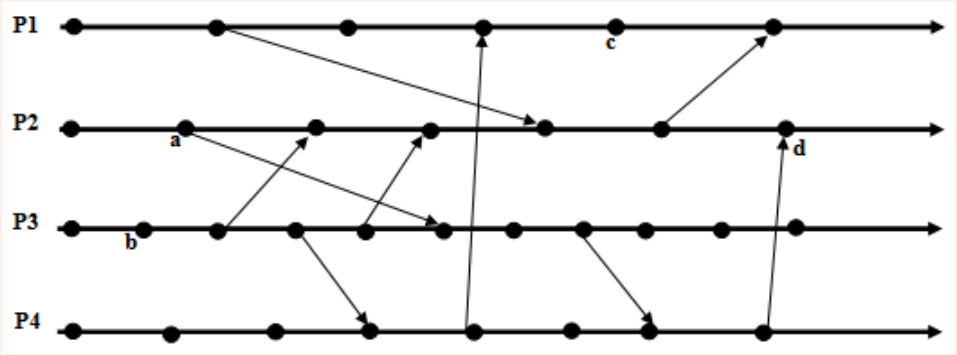


FIGURE 2.12 – Chronogramme des évènements.

1. On considère les événements a, b, c et d de la figure 2.12. Indiquer s'il y à des relations de précédence causales entre ces derniers.
2. Mettre en évidence le non-respect des dépendances causales en émission pour le chronogramme ci-dessus.
3. donner les ensembles d'événements correspondant à :
   - Passé (d)
   - Futur (a)
   - Concurrent (b)

■

Exercice 2.2 (Ordre causal)
On considère l'exécution d'une application répartie représentée dans la figure 2.13.

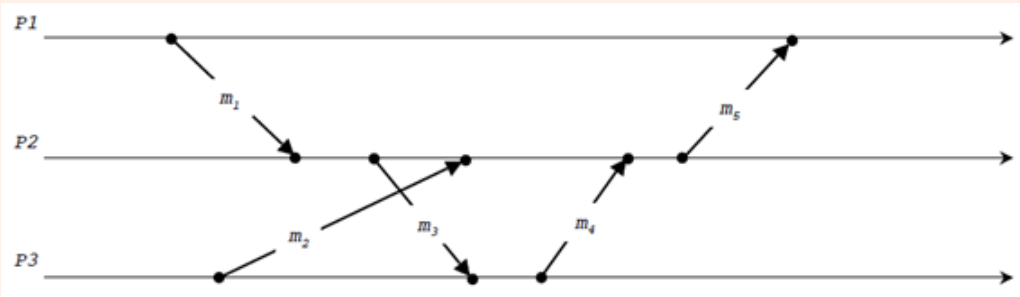


FIGURE 2.13 – Analyse de l'ordre causal.

1. Cette exécution vérifie-t-elle la propriété Causally Ordered (CO) ?
2. justifierez votre réponse en précisant clairement le(s) couple(s) de message(s) que vous considérez.

■

Exercice 2.3 Les exécutions suivantes vérifient-t-elles la propriété Causally Ordered (CO) ?

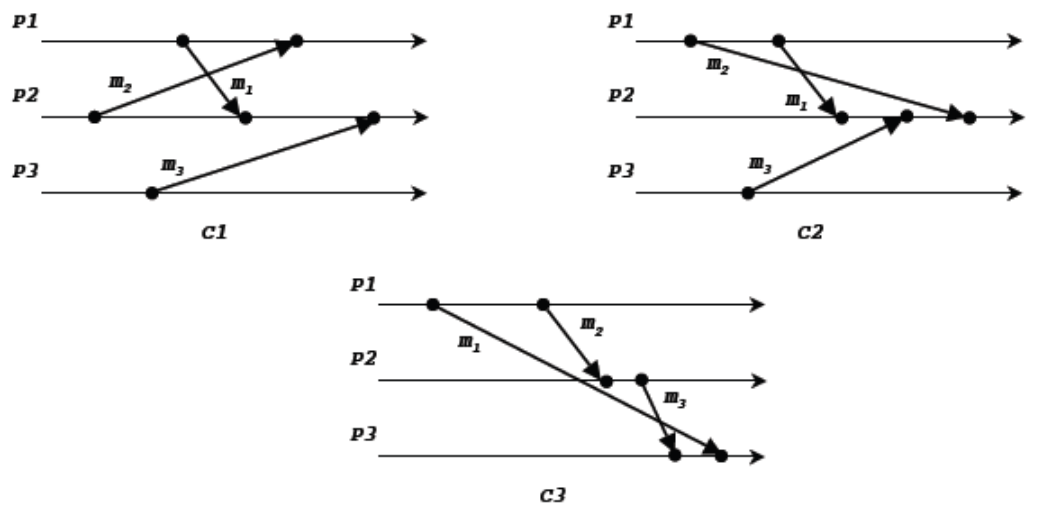


FIGURE 2.14 – Trois exécutions distribuées.

3. Datation des événements

3.1 Introduction

Les systèmes répartis sont présents dans toutes les applications et sont, par nature, très complexes. Le problème principal est qu'il n'y a plus d'état global connu de toutes les parties mais seulement des états locaux permettant de faire émerger un état global.
La propriété d'émergence est souvent mentionnée dans les systèmes à agents ou systèmes multi agents : l'activité de chacun concourt à la réalisation d'un objectif global.
Très naturellement, le contrôle de ces systèmes répartis n'est pas simple mais il est important de pouvoir s'assurer de la réalisation d'un objectif donné par un ensemble d'activités élémentaires.
Un système réparti est constitué de N composants (processus ou sites) communiquant par messages (et uniquement de cette manière).
Chacun de ces composants agit comme un automate: il réalise des opérations qui modifient son état. Les opérations réalisées par un des composants sont naturellement ordonnées par l'ordre dans lequel elles sont réalisées :

— s'il s'agit d'un processus abritant plusieurs activités (threads),
— sur un système monoprocesseur, c'est l'ordre de l'exécution des instructions sur ce processeur qui ordonne les événements.

La définition de l'ordre des événements sur un système multiprocesseurs (fortement et a fortiori faiblement couplés) est plus problématique du fait de la difficulté de maintenir une notion de temps absolu cohérente.

Dans un contexte de répartition:
L'observation des programmes en exécution présente des difficultés qui rendent le problème non trivial.
Un programme réparti est constitué d'un ensemble de processus s'exécutant en parallèle et communiquant seulement par envoi de messages sur des canaux les interconnectant.
Il n'y a pas d'horloge commune aux différents processus, et de plus, les temps de transfert des messages ne sont pas bornés (dans un contexte de communication fiable, ils sont toutefois finis).

Dans ces conditions, il est impossible d'effectuer une observation "simultanée" de l'état des

différents processus et canaux.
La réalisation d'une observation cohérente qui reflète un état global constitué des états locaux des différentes parties du système, pris à des instants physiques différents mais de manière à rendre une information utile sur l'état du système dans son intégralité, constitue donc un problème désormais classique, connu sous le nom de détection d'un état global réparti cohérent [Ma].

3.1.1 Problématique

Obtenir une vision instantanée d'un système réparti, consistant en la collection des états des différents sites le composant (typiquement une image mémoire de chacun des sites) est difficile à obtenir sans figer chacun des systèmes (voir chapitre 4.
L'absence de mémoire commune et le caractère aléatoire des délais d'acheminement des messages échangés entre les sites rendent impossible le calcul d'un état global du système dans un système réparti.
Typiquement, l'image qu'un site possédera de l'état des autres sites ne pourra lui être communiquée qu'au travers de messages et ne pourra de ce fait correspondre qu'à un état du passé de ces sites: la chronologie des différentes images ainsi collectées n'est pas connue a priori.
L'absence d'un état global accessible directement constitue incontestablement une caractéristique de la répartition et est source de difficultés dans le développement d'applications relatives à:

— l'interblocage,ou verrou mortel (deadlock): situation dans laquelle un ensemble de processus est en situation de blocage du fait de l'existence d'un cycle dans le graphe d'allocations et de demandes des ressources à ces processus;
— le ramasse-miettes (garbagecollecting), opération consistant en le épérationdes ressources allouées à un objet inutilisé;
— la mise au point (debugging): opération incluant par exemple la consultation et/ou la modification des valeurs de variables dans différents composants à un instant donné [Ma].

Les Systèmes répartis ont une évolution asynchrone: (i) pas de mémoire commune (communication par messages), (ii) pas d'horloge commune
Les horloges locales ne sont pas synchrones et dérivent
L'état d'un site distant ne peut être connu que par des informations véhiculées par les messages. De plus, les communications introduisent des délais et l'ordre des messages n'est pas forcément préservé, par conséquent:

— Perception différente des mêmes événements depuis des sites distincts
— Chaque site a sa propre horloge, la notion d'état global n'existe pas
— On ne peut pas mettre en oeuvre des algorithmes répartis basés sur le temps.

3.2 Horloges logiques : de Lamport, de Mattern (Vectorielle), Matricielle

3.2.1 Horloge Logique

Lamport [Lam78] a proposé de définir pour ce type de systèmes une notion de temps logique permettant de comparer logiquement des événements du point de vue de leur ordre d'exécution: d'une part, sur un site, les événements locaux peuvent être ordonnés en se basant sur l'ordre de leur exécution (ou le temps absolu s'il est défini) et d'autre part l'émission d'un message sur le site émetteur précède toujours sa réception sur le site récepteur. Cela correspond à la notion de précédence causale [Ben].
Selon Leslie Lamport l'horloge logique permet de comparer logiquement des événements (requêtes, messages...) du point de vue de leur ordre d'exécution (Horloges scalaires)

— Chaque site gère une horloge de type compteur dont la valeur est un entier (initialisé à 0 au lancement du système).

— La valeur de l'horloge logique d'un site est incrémentée chaque fois qu'un événement local s'y produit : opération locale, ou envoi/réception d'un message.
— Dans le cas d'un envoi, la valeur courante (après incrémentation) de l'horloge de l'émetteur est embarquée avec le message et sert à l'estampiller (La réception d'un message permet de synchroniser l'horloge du récepteur avec celle de l'émetteur du message qui est transportée par le message. Le principe est simple : il consiste à attribuer à l'horloge du récepteur une valeur supérieure à la fois à la valeur courante de l'horloge du site et à celle de l'estampille du message reçu.)
— La réception d'un message permet de synchroniser l'horloge du récepteur avec celle de l'émetteur du message qui est transportée par le message.

Construction d'un état global

— L'exécution d'un algorithme réparti est une succession d'événements, chacun d'eux se produisant sur un site donné (Un événement : émission/ réception de message, calcul local au site)
— Une horloge unique permet de donner une date à chacun des événements et de les ordonner entre eux.
— Sur chaque site, il est possible de définir un état local et de définir un ordre entre les événements
— Deux processus de deux sites différents peuvent avoir des informations différentes sur l'état du système et sur l'ordre des événements qui s'y produisent.
— La solution de synchronisation des sites entre eux est donnée par la construction d'un état global
— Utilisation d'horloges : logiques, vectorielles, matricielles
— Utilisation d'états locaux des sites et de messages en transit entre eux

Temps logique

Un temps logique
— Temps qui n'est pas lié à un temps physique
— But est de pouvoir préciser l'ordonnancement de l'exécution des processus et de leur communication
— En fonction des événements locaux des processus, des messages envoyés et reçus, on créé un ordonnancement logique
— Ordonnancement des événements
— Les dépendances causales définissent des ordres partiels pour des ensembles d'événements
— Buts d'une horloge logique (selon le type d'horloge)
— Créer un ordre total global sur tous les événements de tous les processus
— Déterminer si un événement a eu lieu avant un autre ou s'il n'y a pas de dépendances causales entre eux
— Vérifier que des propriétés d'ordre sur l'arrivée de messages sont respectées

Principe: datation de chacun des événements du système avec respect des dépendances causales entre événements. Il existe 3 familles d'horloge :
— Estampille (horloge de Lamport) : une donnée par événement.
— Vectorielle (horloge de Mattern) : un vecteur par événement.
— Matricielle : une matrice par événement.

3.2.2 HORLOGE LOGIQUE SCALAIRE

Introduit en 1978 par Leslie Lamport. C'est le premier type d'horloge logique introduit en informatique.

Une date (estampille) est associée à chaque événement. Une estampille représente un couple

(i, nb).
i : numéro du processus.
nb : numéro d'événement.
Principe: attribuer à l'horloge du récepteur une valeur supérieure à la fois à a valeur courante de l'horloge du site et à celle de l'estampille du message reçu.
Conséquence: Garantit que la réception sera postérieure à l'émission.

Création du temps logique

- Chaque site i gère un compteur (une horloge HL_i) dont la valeur est un entier, initialisée à 0.
- chaque message m envoyé est estampillée (daté) (EL_m) par la date de son évènement.
  - Si un évènement est local, alors $HL_i \leftarrow HL_i + 1$
  - Si l'évènement est l' envoi d'un message m alors

$$\begin{cases} HL_i \leftarrow HL_i + 1 \\ EL_m \leftarrow HL_i \end{cases}$$

- Si l'évènement est la réception du message m alors

$$HLi \leftarrow max(HL_i, EL_m) + 1$$

■ **Exemple 3.1** Créer un temps logique à l'aide de l'horloge de Lamport.

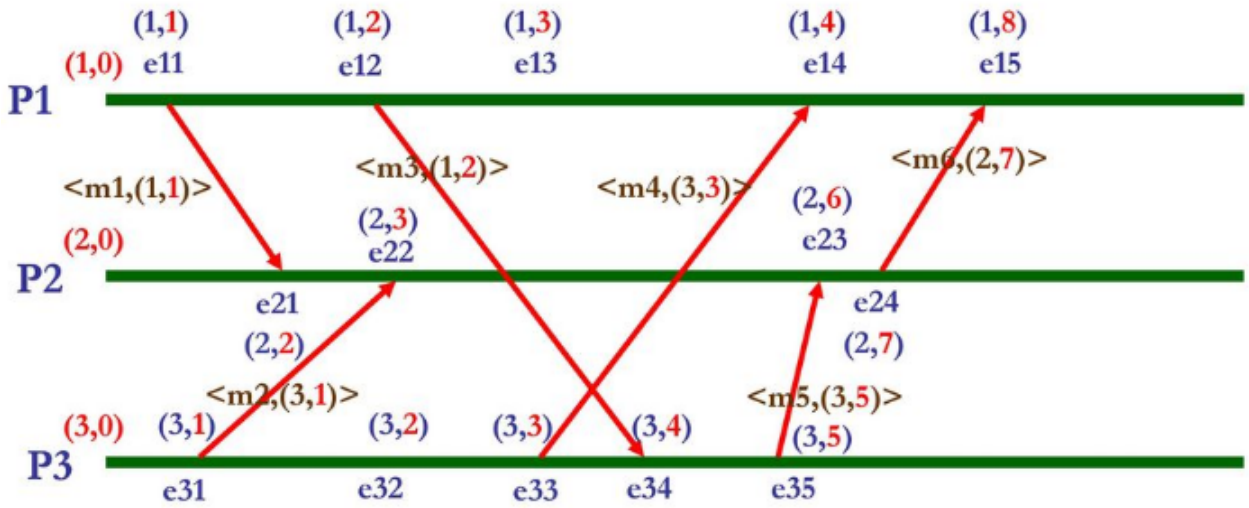


FIGURE 3.1 – Datation horloge scalaire.

Pour $e11, e12, e13...$: incrémentation de +1 de l'horloge locale.
Date de $e23$: 6 car le message $m5$ reçu avait une valeur de 5 et l'horloge locale est seulement à 3 $(max(3,5)+1)$.
Date de $e34$: 4 car on incrémente l'horloge locale vu que sa valeur est supérieure à celle du message $m3(max(3,2)+1)$.

L'ordre des évènements n'est pas un ordre strict : plusieurs évènements peuvent porter la même valeur. ■

■ **Exemple 3.2** Chronogramme avec ajout des estampilles.

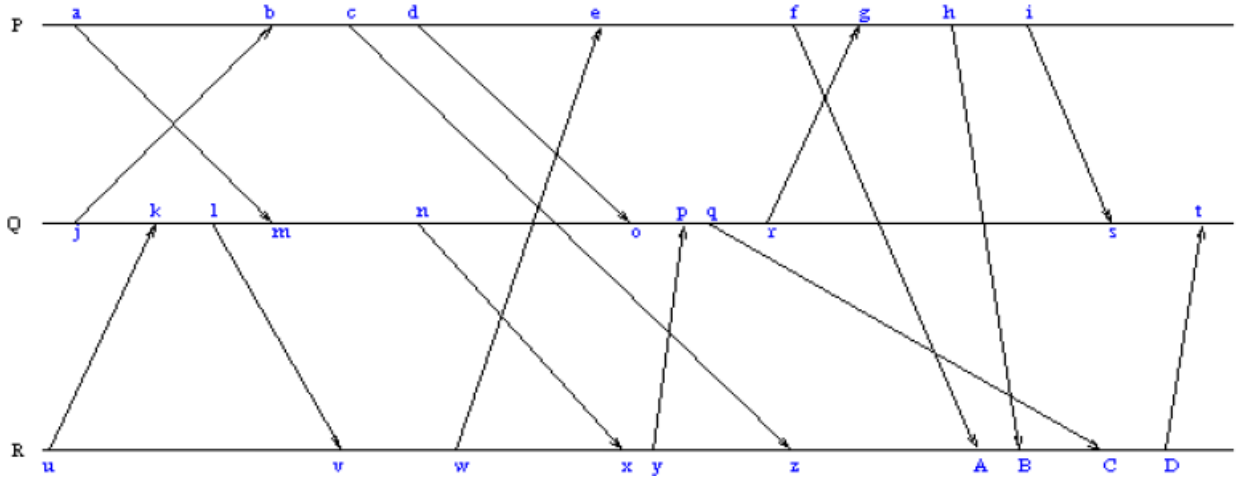


FIGURE 3.2 – Chronogramme des évènements.

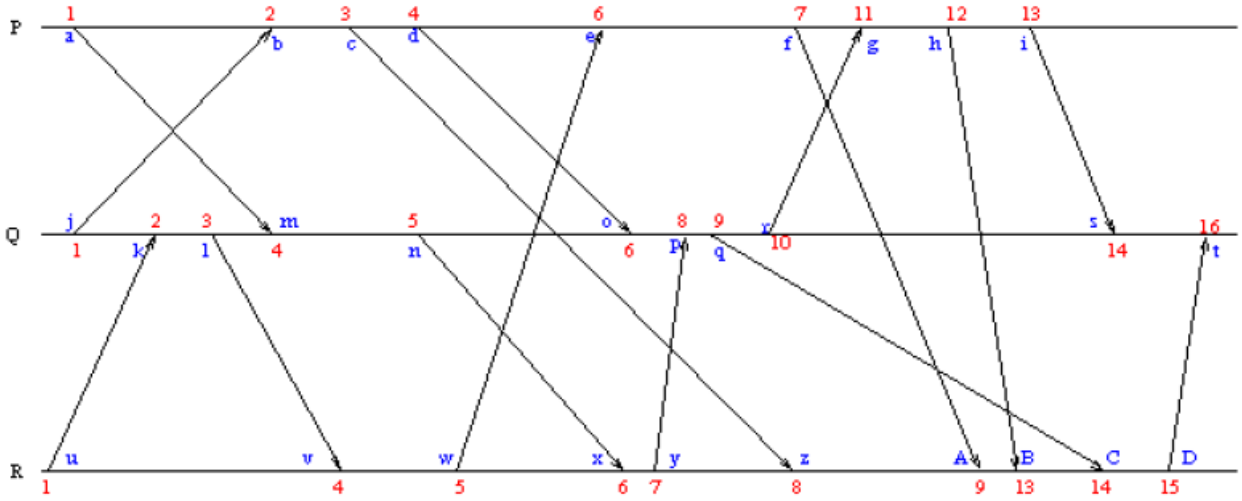


FIGURE 3.3 – Datation horloge scalaire.

■

Cette technique permet donc d'associer à chaque événement une date (estampille logique) correspondant à la valeur de l'horloge de son site modifiée selon les règles que nous venons de définir. On peut observer que:

— l'ordre des événements qui est ainsi défini n'est pas un ordre strict: plusieurs événements peuvent porter la même valeur. C'est le cas (parmi d'autres) sur notre exemple (figure 3.3 des événements e, o et x appartenant respectivement à P, Q et R qui ont chacun 6 comme date. Il est facile de rendre cet ordre strict en modifiant légèrement le système de datation : la date d'un événement sur un site est obtenue en adjoignant à la valeur de l'horloge scalaire de ce site, l'identification du site (par exemple un entier attribué artificiellement ou une adresse IP ou physique).

Détection de la causalité à base d'horloges scalaires
Propriété:

Propriété: L'horloge de Lamport respecte la dépendance causale :

$$e \rightarrow e' \Rightarrow (HL(e) < HL(e'))$$

Ordonnancement global:

Via HL_i, on ordonne tous les événements du système entre eux. Ordre total, noté $e << e'$: e s'est déroulé avant e'. Soit e événement de P_i et e' événement de P_j :

$$e << e' \Leftrightarrow (HL_i(e) < HL_j(e')) \; ou \; (HL_i(e) = HL_j(e') \; avec \; i < j)$$

Localement (si $i = j$), HL_i donne l'ordre des événements du processus.
Les 2 horloges de 2 processus différents permettent de déterminer l'ordonnancement des événements des 2 processus. Si égalité de la valeur de l'horloge, le numéro du processus est utilisé pour les ordonner.

— l'estampille logique HL(e) de e sur le site i est un couple (HL_i, i).
— L'ordre sur les estampilles est le suivant :

$$(HL_i, i) < (HL_j, j) \; si \; et \; seulement \; si$$

$$\begin{cases} HL_i < HL_j \\ ou \; HL_i = HL_j \wedge i < j \end{cases}$$

■ **Exemple 3.3** Ordre total global obtenu pour d'exemple de la figure 3.3.

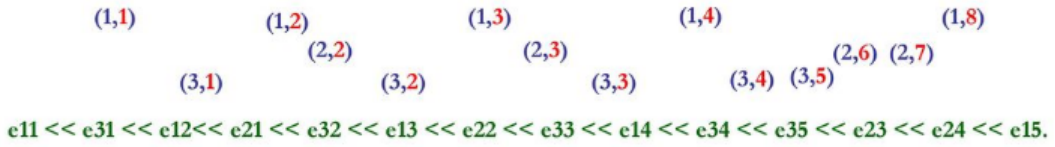


FIGURE 3.4 – Ordre total global obtenu

■

Pour résumé : L'horloge de Lamport respecte la dépendance causale :

$$e \rightarrow e' \Rightarrow (HL(e) < HL(e'))$$

Mais pas la réciproque :

$$(HL(e) < HL(e')) \not\Rightarrow e \rightarrow e'$$

Utilité de l'horloge de Lamport: Faire l'ordonnancement global des événements dans un système distribué.

3.2.3 HORLOGE LOGIQUE VECTORIELLE

Nous venons de voir que le système de datation par estampilles scalaires d'une part introduisait un ordre artificiel sur des événements concurrents et d'autre part ne permettait pas de corriger les défaillances vis-à-vis de la relation FIFO des canaux de communication. Le mécanisme de datation par estampilles vectorielles (et les horloges vectorielles maintenues par les différents composants

d'un système) permet de pallier ces deux inconvénients. L'horloge vectorielle a été introduite indépendamment en 1988 par Colin Fidge et Friedemann Mattern.

Chaque site gère une horloge vectorielle constituée de n entiers (n est le nombre de composants du système). Une telle horloge permet de dater les événements d'un site et est mise à jour lors de l'occurrence des événements. Comme pour les horloges scalaires, les messages envoyés par un site sont estampillés en utilisant la valeur courante de l'horloge vectorielle du site émetteur et la réception d'un message permet au site récepteur de synchroniser son horloge vectorielle avec celle du site émetteur du message.

— Horloge qui assure la réciproque de la dépendance causale : $H(e) < H(e') \Rightarrow (e \rightarrow e')$.
— Permet également de savoir si 2 événements sont parallèles (non dépendants causalement).
— Ne définit par contre pas un ordre total global.

Fonctionnement de l'horloge

— Chaque site i gère un vecteur d'entiers de n éléments (une horloge HV_i) avec n le nombre de sites
— Chaque message m envoyé est estampillé (daté) (EV_m) par la date de son évènement.
  — Si un évènement est locale, alors

$$HV_i[i] \leftarrow HL_i[i] + 1$$

  — Si l'évènement est l'envoi d'un message m alors

$$HVi[i] \leftarrow HL_i[i] + 1$$

$$EV_m \leftarrow HV_i$$

  — Si l'évènement est la réception du message m alors [Coh]

$$\begin{cases} HV_i[i] \leftarrow HL_i[i] + 1 \\ HL_i[j] \leftarrow max(HL_i[j], EL_m[j]) \ pour\ tout\ j \neq i \end{cases}$$

Principe :

— Utilisation de vecteur HV de taille égale au nombre de processus.
— Localement, chaque processus P_i a un vecteur HV_i.
— Un message est envoyé avec un vecteur de date.
— Pour chaque processus P_i, chaque case $HV_i[j]$ du vecteur contiendra des valeurs de l'horloge du processus P_j.

■ **Exemple 3.4** Même exemple que pour l'horloge scalaire.

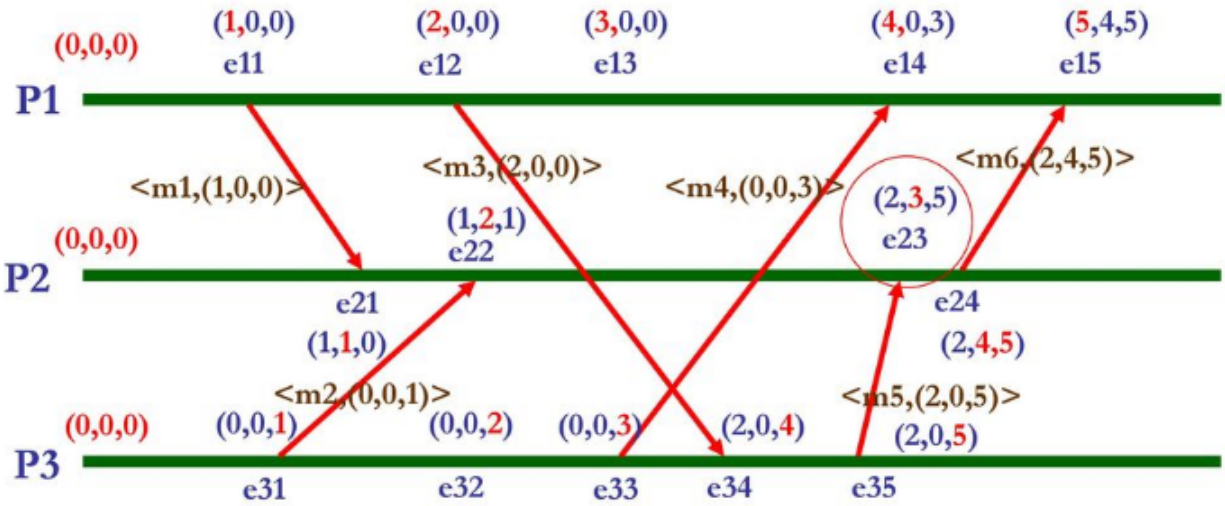


FIGURE 3.5 – Datation horloge vectorielle.

$V(e23) = (2,3,5)$: (2 relatif à $P1$, 3 à $P2$, 5 à $P3$).

— $3 = 2+1$ (Incrémentation du compteur local). 3ème évent dans $P2$.

— $2 = max(2,0)$. 2 évents dans $P1$ ($e11$ et $e12$) qui sont en dépendance causale par rapport à l'évent $e23$.

— $5 = max(5,3)$. 5 évents dans $P3(e31, e32, e33, e34$ et $e35)$ qui sont en dépendance causale par rapport à l'évent $e23$.

■

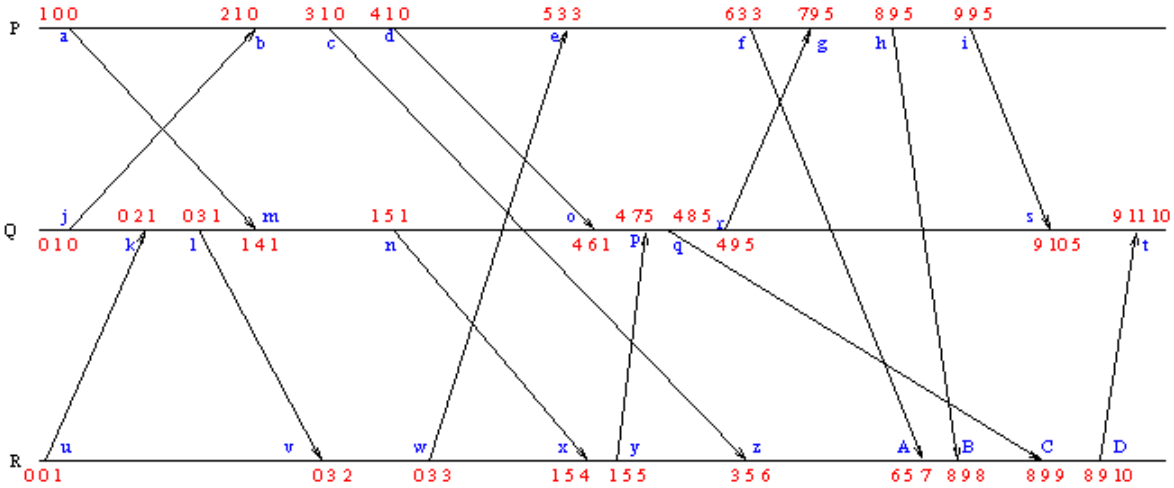


FIGURE 3.6 – Datation horloge vectorielle.

Propriété :

la valeur de la i^{me} composante de HVe correspond au nombre d'évènements du site S_i appartenant au passé de e.

$$\forall i, HV_e[i] = card(e' : e' \in Si \wedge e' \rightsquigarrow e)$$

■ **Exemple 3.5** Par exemple de la figure 3.6 l'estampille vectorielle de l'événement p est $[4,7,5]$.

Cela correspond au fait que Passé (p) contient
- 4 événements sur $P(a,b,c,d)$;
- 7 événements sur $Q(j,k,l,m,n,o,p)$;
- 5 événements sur $R(u,v,w,x,z)$.

■

La relation d'ordre:

Relation d'ordre partiel sur les dates
- $HV \leq HV'$ défini par $\forall i : HV[i] \leq HV'[i]$
- $HV < HV'$ défini par $HV \leq HV'$ *et* $\exists j$ *tel que* $HV[j] < HV'[j]$
- $HV||HV'$ défini par $\neg(HV < HV') \wedge \neg(HV' < HV)$

Dépendance et indépendance causales L'horloge de Mattern assure les propriétés suivantes, avec e et e' deux événements et $HV(e)$ et $HV(e')$ leurs datations
- $HV(e) < HV(e') \Rightarrow e \rightarrow e'$
- Si deux dates sont ordonnées, on a forcément une dépendance causale entre les événements datés
- $HV(e)||HV(e') \Rightarrow e||e'$
- Si il n'y a aucun ordre entre les 2 dates, les 2 événements sont indépendants causalement

■ **Exemple 3.6** Par exemple $[4,7,5]$ est plus petite que $[6,7,8]$, plus grande que $[4,6,4]$ et incomparable avec $[6,5,7]$

On peut vérifier sur notre exemple de la figure 3.6 que :
- les estampilles vectorielles des événements précédant causalement **p** sont inférieures (au sens qui a été défini) à l'estampille vectorielle de $p([4,7,5])$. Par exemple l'estampille vectorielle de l'événement **d** antérieur à **p** est $([4,1,0])$ qui est inférieure à $([4,7,5])$;
- les estampilles vectorielles des événements suivant causalement **p** sont supérieures à l'estampille vectorielle de **p**. Par exemple l'estampille vectorielle de l'événement **C** postérieur à **p** est $([8,9,9])$ qui est supérieure à $([4,7,5])$;
- les estampilles vectorielles des événements concurrents de **p** sont incomparables avec l'estampille vectorielle de **p**; Par exemple l'estampille vectorielle de l'événement **e** concurrent avec l'événement **p** est $([5,3,3])$ qui est incomparable avec $([4,7,5])$.

■

Limite de l'horloge de Mattern
- Ne permet pas de définir un ordre global total
- En cas de nombreux processus, la taille du vecteur peut-être importante et donc des données à transmettre relativement importante

3.2.4 HORLOGE LOGIQUE MATRICIELLE

Horloges et estampilles matricielles
- Chaque site i gère une horloge (notée HM_i) matricielle (nn) avec n le nombre de sites
- Chaque message m envoyé est estampillé (daté) (EM_m) par la valeur courante de HM_i .
- Ligne i : informations sur événements de P_i :
  - $HM_i[i,i]$: nombre d'événements réalisés par P_i.
  - $HM_i[i,j]$: nombre de messages envoyés par P_i à P_j (avec $j \neq i$).
- Que signifie $HM_i[j,k]$ (Ligne j (avec $j \neq i$))?
  - $HM_i[j,k]$ = nbre de messages issus de p_j vers p_k dont p_i a connaissance (avec $j \neq k$)
  - $HM_i[j,j]$: correspond au nbre d'évènements locaux du site j que l'on connaît sur P_j.

Avec 3 processus :

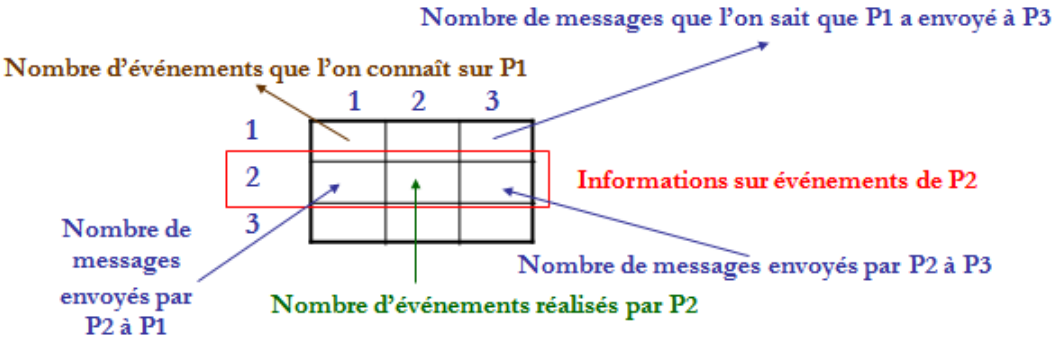


FIGURE 3.7 – Horloge Matricielle.

Comment HM_i est-elle modifiée ?

— Évènement local à p_i : $HM_i[i,i] \rightarrow HM_i[i,i]+1$

— Émission de m de p_i vers p_j :

$$\begin{cases} HM_i[i,i] \rightarrow HM_i[i,i]+1 \\ HMi[i,j] \rightarrow HMi[i,j]+1 \\ m\ est\ estampill\ :\ EM_m = HM_i \end{cases}$$

— Réception de (m, EM_m) en provenance p_j :

— Quand un message peut-il être délivré ? quand tous les messages antérieurs à lui ont été délivrés, i.e :

— Respect de l'ordre FIFO sur le canal $j \leftarrow i$:

$$EMm[j,i] = HMi[j,i]$$

— Respect de l'ordre de réception :

$$EM_m[k,i] = HMi[k,i], \forall k \neq i$$

— Lors de la délivrance du message (m, EM_m) en provenance de p_j , mettre à jour l'horloge

$$\begin{cases} HM_i[i,i] \rightarrow HM_i[i,i]+1 \\ pour\ tout\ k \neq i\ pour\ tout\ l \neq i : \\ HM_i[k,l] \rightarrow max(HM_i[k,l], EM_m[k,l]) \end{cases}$$

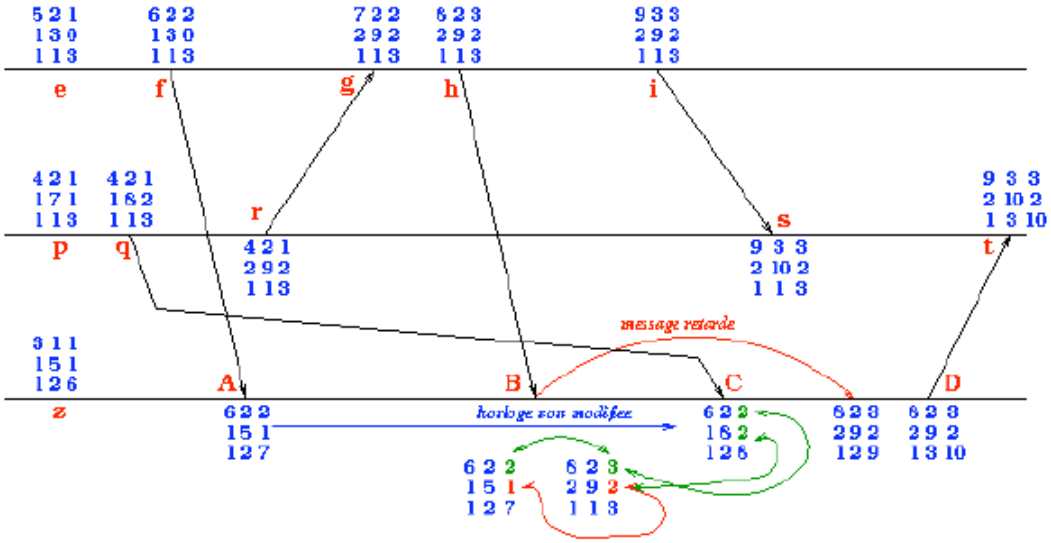


FIGURE 3.8 – Datation avec l'horloge matricielle

■ **Exemple 3.7** ■

Utilité de l'horloge matricielle

Assurer la délivrance causale de messages entre plusieurs processus.

3.3 Conclusion

— Horloge scalaire :
HL_i : ce que p_i connaît du système (nbre d'évènements)
— Horloge vectorielle :
$HV_i[j]$:ce que p_i connaît du site p_j
— Horloge matricielle :
$HM_i[j,k]$:ce que p_i connaît de la connaissance du site p_j sur le site p_k

3.4 Exercices

Exercice 3.1 On considère un système distribué 2.12 constitué de 4 sites P_1, P_2, P_3, s'envoyant des messages de façon asynchrone comme représenté par la figure suivante.

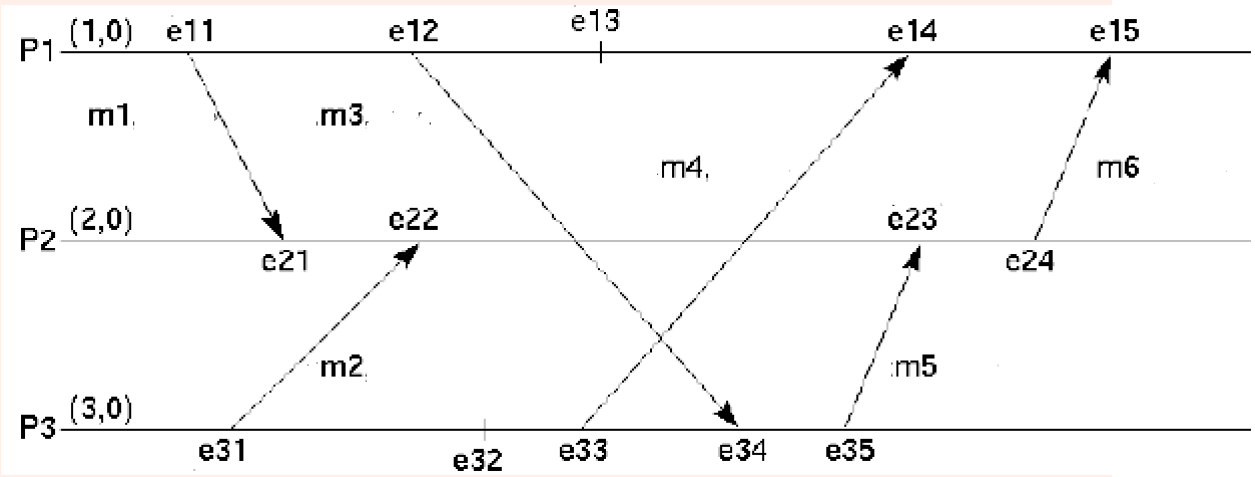


FIGURE 3.9 – Chronogramme des évènements.

1. Indiquez au dessus de chaque point de la figure 3.9, la valeur de l'horloge scalaire et vectorielle du processus où se produit l'événement correspondant.
2. Indiquer l'ordre total obtenu.

■

Exercice 3.2 L'objectif est de comparer deux évènements e_1 et e_2 qui se passent dans deux sites différents. Nous supposerons que l'évènement e_1 (resp. e_2) est un évènement local du site 1 (resp. 2).

— Supposons que les deux sites ont une horloge de Lamport comme système de datation. Comparer les deux évènements e_1 et e_2 dans les deux situations suivantes :

1. la date de l'évènement e_1 est 4 et celle de e_2 est 3.
2. la date de l'évènement e_1 est 4 et celle de e_2 est 4.

— Supposons que les deux sites ont une horloge vectorielle comme système de datation et que le système distribué est composé de trois sites. Comparer les deux évènements e_1 et e_2 dans les deux situations suivantes

1. la date de l'évènement e_1 est $\begin{pmatrix} 4 \\ 3 \\ 2 \end{pmatrix}$ et celle de e_2 est $\begin{pmatrix} 5 \\ 2 \\ 1 \end{pmatrix}$.
2. la date de l'évènement e_1 est $\begin{pmatrix} 5 \\ 2 \\ 4 \end{pmatrix}$ et celle de e_2 est $\begin{pmatrix} 4 \\ 3 \\ 6 \end{pmatrix}$.

■

Exercice 3.3 On considère maintenant un système distribué constitué de trois sites nommés $P1$,$P2$ et $P3$, utilisant des horloges matricielles pour dater les évènements de chaque processus.

Après quelques instants d'exécution, les horloges des processus $P1, P2$ et $P3$ indiquent les

dates suivantes : $HM_3 = \begin{pmatrix} 3 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 2 \end{pmatrix}$ $HM_2 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix}$ $HM_3 = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{pmatrix}$

- Au total, combien de messages ont été échangés ?
- Au total, combien d'évènement ont eu lieu ?
- Au total, combien d'évènement locaux ont eu lieu ? (sans compter les évènements des émissions ou des réceptions, ni l'état initial)
- Dans quel ordre ont été envoyé les messages de $P2$? Justifiez

4. État global d'un Systèmes Répartis

4.1 Définition, État d'une coupe et datation d'un état

Définition 4.1.1 État global : état du système à un instant donné Défini à partir de coupures.
Buts de la recherche d'états globaux :

- Trouver des états cohérents à partir desquels on peut reprendre un calcul distribué en cas de plantage du système.
- Détection de propriétés stables, du respect d'invariants
- Faciliter le debugging et la mise au point d'applications distribuées

Définition 4.1.2 Coupure : photographie à un instant donné de l'état du système. Définit les événements appartenant au passé et au futur par rapport à l'instant de la coupure.

C'est une capture de l'état du dernier évènement avant "la photo" sur chaque site.

Calcul distribué = ensemble E d'événements. Coupure C est un sous-ensemble fini de E tel que : Soit a et b deux événements du même processus :

$$((a \in C) et (b \leftarrow a)) \Rightarrow (b \in C).$$

Si un événement d'un processus appartient à la coupure, alors tous les événements locaux le précédant y appartiennent également.

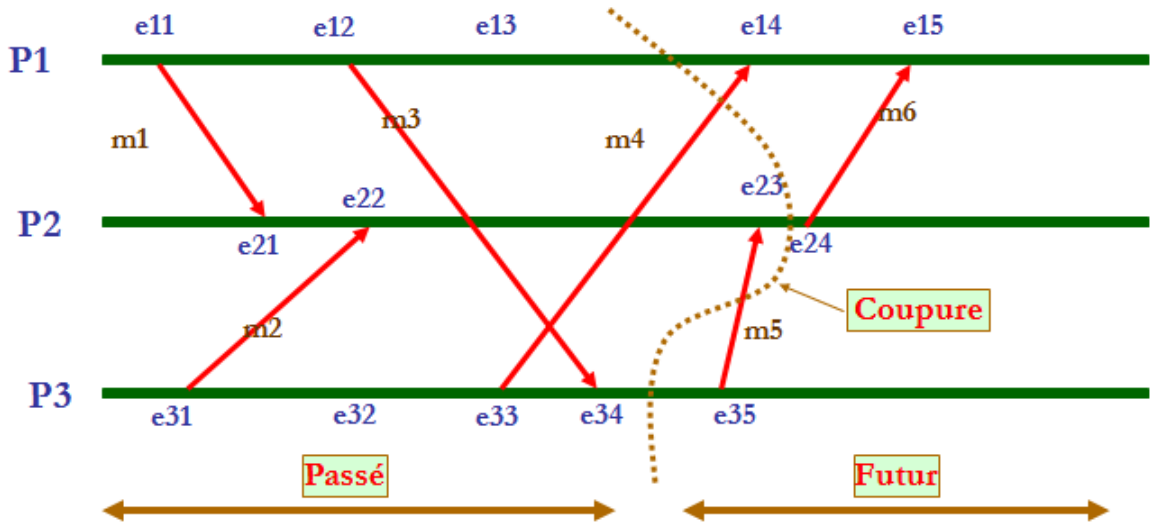


FIGURE 4.1 – Exemple de coupure

■ **Exemple 4.1** Coupure = ensemble $\{e11, e12, e13, e21, e22, e23, e31, e32, e33, e34\}$.

■

4.1.1 État associé à une coupure

Si le système est composé de N processus, l'état associé à une coupure est défini au niveau d'un ensemble de N événements $(e_1, e_2, ...e_i, ...e_N)$, avec e_i événement du processus P_i tel que :

$$\forall i, \forall e \in C : (e \text{ évènement du processus } P_i) \Rightarrow (e \leftarrow e_i)$$

L'état est défini à la frontière de la coupure : l'événement le plus récent pour chaque processus.

■ **Exemple 4.2** Suivant l'exemple de la figure 4.1

État de la coupure $= (e13, e23, e34)$.

■

4.2 Capture d'une coupure cohérente

4.2.1 Coupure cohérente

Définition 4.2.1 Coupure cohérente : coupure qui respecte les dépendances causales des événements du système et pas seulement les dépendances causales locales à chaque processus.

Soit a et b deux événements du système :

$$((a \in C) et (b \rightarrow a)) \Rightarrow (b \in C)$$

Coupure cohérente : aucun message ne vient du futur.

État cohérent : État associé à une coupure cohérente. Permet par exemple une reprise sur faute.

■ **Exemple 4.3** Suivant l'exemple de la figure 4.1
Coupure $C1$: non cohérente car : $e23 \in C1$ et $e35 \rightarrow e23$ mais $e35 \notin C1$. La réception de $m5$ est dans la coupure mais pas son émission. $m5$ vient du futur par rapport à la coupure.

■

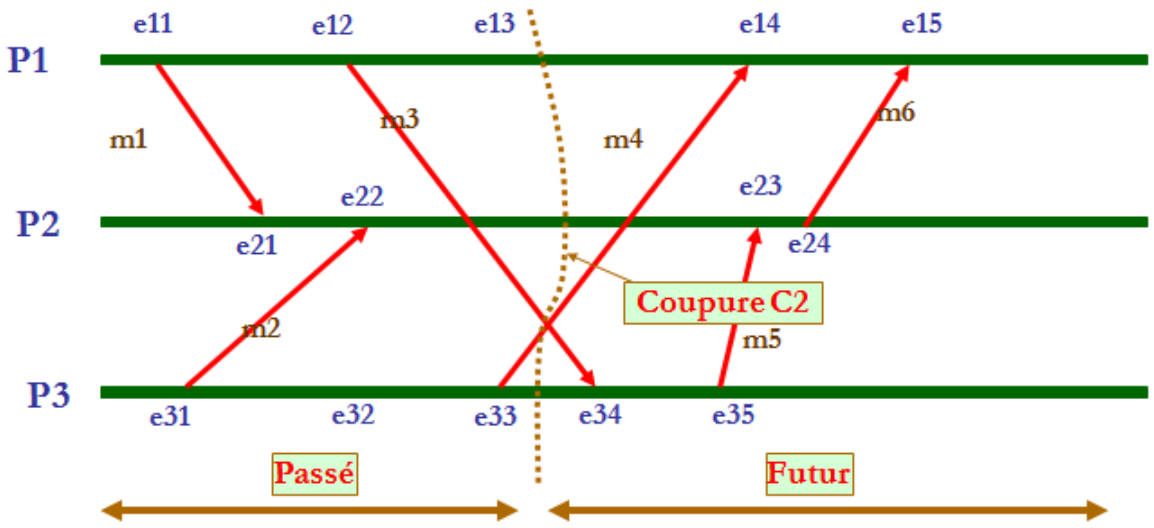


FIGURE 4.2 – Exemple de coupure

■ **Exemple 4.4** Coupure $C2$: cohérente.

■

4.2.2 Caractérisation des coupures cohérentes

Soit c_i les évènements définissant la coupure : $C = (c_1, ..., c_n)$

Propriété : $(\forall i, j, c_j || c_i) \Leftrightarrow C$ est une coupure cohérente.

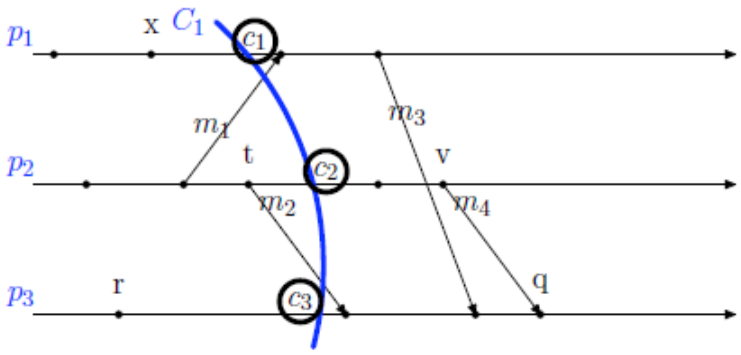


FIGURE 4.3 – $C1 = (c_1, c_2, c_3)$

■ **Exemple 4.5** $C_1 = (x, t, r)$

$$x||t, x||r, t||r \Rightarrow C_1 \textit{ est une coupure cohérente}$$

■

4.3 Datation de Coupure

4.3.1 estampillage vectorielle

On considère la datation vectorielle des évènements et on associe à une coupure (définie par les évènements $< e_1, e_2, ..., e_N >$ / e_j est l'évènement le plus récent du site S_j appartenant à la

coupure).
Associons à la coupure $C = (c_1, ..., c_n)$ l'estampille vectorielle $HV(C)$ défini par:

$$HV(C) = sup(HV(c_1), ..., HV(c_n))$$

en d'autres termes:

$$HV(C) = max(HV(e1), ..., HV(eN)) :$$

$$\forall i : HV(C)[i] = max(V(e1)[i], ..., V(e_N)[i])$$

Pour chaque valeur du vecteur, on prend le maximum des valeurs de tous les vecteurs des N événements pour le même indice.

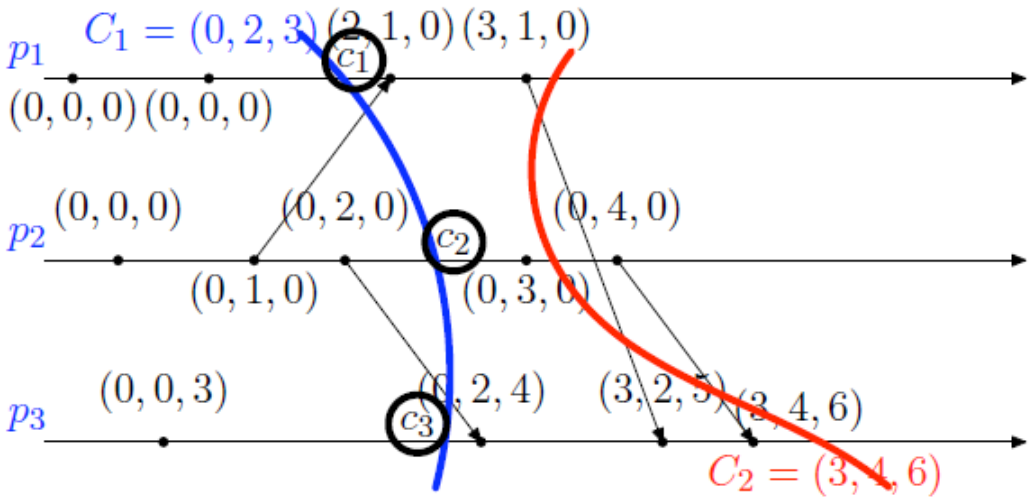


FIGURE 4.4 – Dates des coupure C1 et C2

■ **Exemple 4.6** $C_1 = (x,t,r)$

$$HV(C_1) = sup(HV(x), HV(t), HV(r)) = sup((2,0,0),(0,3,0),(0,0,1)) = (2,3,1)$$

■

4.3.2 Condition pour que les coupures soient cohérentes

Propriété : C cohérente si et seulement si:

$$HV(C) = (HV(c_1)[1], ..., HV(c_i)[i], ..., HV(c_N)[N])$$

$$HV(C) = (HV(e_1)[1], ..., HV(e_i)[i], ..., HV(e_N)[N])$$

■ **Exemple 4.7** Même exemple de la figure 4.4

$C_1 = (x,t,r)$

$$HV(C_1) = sup(HV(x), HV(t), HV(r)) \tag{4.1}$$
$$= sup((2,0,0),(0,3,0),(0,0,1)) \tag{4.2}$$
$$= (2,3,1) \tag{4.3}$$
$$= ((2,0,0)[1],(0,3,0)[2],(0,0,1)[3]) \tag{4.4}$$

donc C_1 est une coupure cohérente.

$C_2 = (x,t,r)$

$$HV(C_2) = sup((4,2,0),(0,4,0),(4,5,4)) = (4,5,4)$$

C_2 est une coupure non cohérente car $(4,2,5) \neq (4,4,4)$

■

4.4 État du système

Motivation:

— Observation, détection de propriétés.

— reprise en cas de panne

Contraintes:

— L' état enregistré doit être cohérent

— Le coût de l'enregistrement doit être raisonnable

Hypothèses:

— les canaux de communication entre processus sont FIFO

— Un seul processus décide de lancer la procédure d'enregistrement

Objectif: enregistrement d'un état cohérent en un temps fini. sous quelle forme ?

Définition 4.4.1 État

L'état d'un système réparti est:

— l'ensemble des états de chacun de ses sites

— et l'ensemble des états de chacun de ses canaux de communications

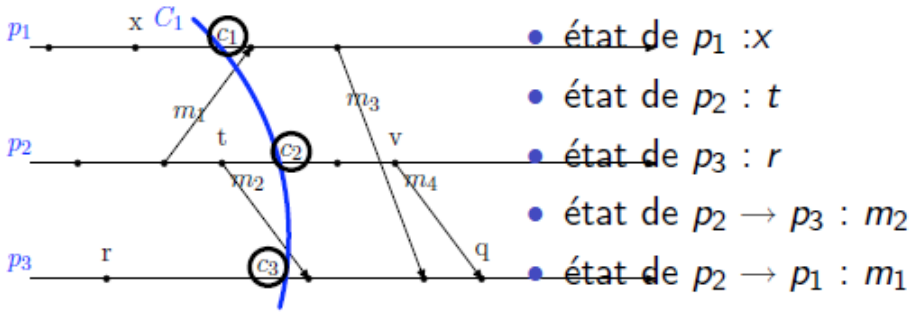


FIGURE 4.5 – État du système

■ **Exemple 4.8** ■

4.5 Exercices

Exercice 4.1 On considère les deux coupures désignées $C1$ et $C2$ dans la figure 4.6 ci-dessous :

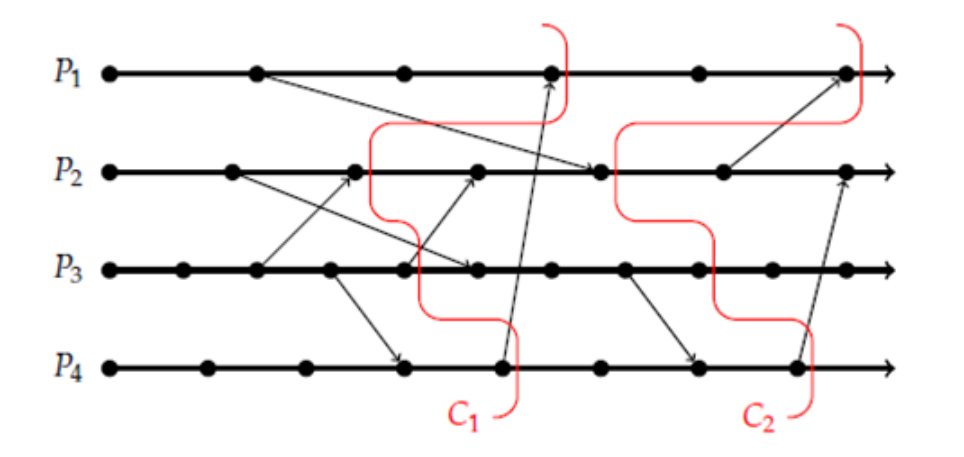


FIGURE 4.6 – Deux coupures

1. Les coupures C1 et C2 sont elles cohérentes? Justifier vos réponses (n'hésitez pas à nommer des états sur la figure pour vous aider à vous justifier)
2. Quelle est l'état du système au cours de la coupure C1?
3. Supposons que l'état de chaque processus ait été sauvegardé au moment de la coupure. Les valeurs des horloges de Lamport sauvegardées avec chaque processus suffisent-elles pour décider si la combinaison des ces états locaux forme un état global cohérent ? Comment peut-on détecter ces incohérences autrement?
4. Dater les événements avant la coupure C1 en utilisant les horloges vectorielles et donner la date de la coupure C1.

5. Réplication des données dans un SR

5.1 Définition, Caractéristiques, Avantages et inconvénients

5.1.1 Réplication

L'idée d'utiliser la redondance dans les systèmes informatiques pour masquer les défaillances des composants a été introduite par Von Neumann. Avec plusieurs répliques, une entité répliquée continue à fournir un service à un client même si une ou plusieurs répliques sont défaillantes. Dans un système informatique, la redondance peut être utilisée au niveau du stockage des données, des ressources de calcul, des liens de communication entre client et serveur, ou encore au niveau des composants de l'application elle même.

La gestion des données est une des principales raisons d'être de l'informatique. Avec la démocratisation de l'Internet, la popularisation des périphériques tels les smartphones, les appareils photos et les caméscopes numériques, le nombre de sources de données numériques est très grand. De plus, de nombreuses applications, dans le domaine scientifique ou dans le monde de la finance notamment, génèrent de grandes masses de données. La croissance exponentielle des données générées avec l'avènement de l'ère du Big Data crée de nouveaux défis, à la fois pour leur stockage et leur traitement. Les systèmes de gestion de données se doivent d'offrir un stockage fiable, performant et garantissant un certain niveau de cohérence. Pour cela, la réplication est une technique clé.

5.1.2 La réplication

La réplication est le processus de création et de placement de copies d'entités logicielles ou matérielles. Les entités dupliquées peuvent être des données, du code ou les deux à la fois. Des exemples sont respectivement les fichiers, les tâches de calcul scientifique et les objets.

La création de copies consiste à reproduire la structure et l'état des entités dupliquées. La copie d'un fichier est un autre fichier de même contenu. La copie d'un programme est un autre programme qui exécute le même code et dont l'état d'exécution est celui du programme initial. Le placement des copies consiste à choisir, en fonction des objectifs de réplication, un environnement d'exécution pour les copies. Par exemple, le placement dans un environnement accessible localement assure le travail en mode déconnecté alors que le placement sur plusieurs machines permet la distribution

de la charge. Le processus de réplication peut être décomposé en plusieurs opérations. La mise en oeuvre de ces opérations est définie dans un protocole de réplication[Abi07].

Une tâche dupliquée : est une tâche qui possède *n* exemplaires afin de recouvrir les erreurs (*n* représente le niveau de réplication). Chaque exemplaire d'une tâche dupliquée est appelé une copie et est placé sur un calculateur distinct [**[Abi07]**].

5.1.3 Fiabilité

Afin d'éviter de perdre définitivement une donnée en cas de crash disque (pérennité) ou de permettre l'accès même lorsque certaines machines sont déconnectées (disponibilité), il est nécessaire d'ajouter de la redondance. Il existe pour cela deux grandes familles de solutions : (i) la réplication et (ii) les codes correcteurs. Le principe de base est simple : disposer de plusieurs copies d'une même donnée de manière à ce que celle-ci soit toujours disponible. Cependant, si l'on prend en compte la disposition des copies (affinités de données/fautes corrélées), l'efficacité des mécanismes de réparation en cas de faute, la répartition de la charge sur les nœuds et sur le réseau, la conception de mécanismes de réplication s'avère complexe.

5.1.4 Performance

La réplication des données est également utile pour offrir de bonnes performances d'accès. En effet, elle permet de rapprocher des copies des utilisateurs pour diminuer la latence, ou de créer de nombreuses copies pour des données extrêmement populaires afin de répartir la charge sur différents serveurs. En revanche, la maintenance des copies d'une donnée peut s'avérer coûteuse et dégrader les performances, surtout lorsque la donnée est fréquemment mise à jour. Il est donc important d'adapter la réplication des données en fonction des schémas d'accès, et de localiser efficacement les copies des données.

5.1.5 La cohérence

La cohérence est une relation qui définit le degré de similitude entre les copies d'une entité dupliquée. Dans le cas idéal, cette relation caractérise des copies qui ont des comportements identiques. Dans les cas réels, où les copies évoluent de manière différente, la cohérence définit les limites de divergence autorisées entre les copies.

La tolérance aux pannes est une des utilisations principales de la réplication et de la cohérence. Les pannes de machines sont tolérées en copiant un service sur plusieurs machines, alors que les pannes de connexion sont gérées en copiant les services sur des machines accessibles par des chemins différents. Les pannes de connexion sont également gérées dans les environnements à usagers mobiles où la réplication et la cohérence permettent le travail en mode déconnecté. La distinction, par exemple, entre "réplication active" et "réplication passive" ne porte pas sur le processus de réplication mais sur la gestion de la cohérence entre copies. L'appellation s'adresse, en effet, non aux copies mais aux échanges entre elles[**ABID**].

Maintenir plusieurs copies d'une même donnée peut poser des problèmes de cohérence lors des mises à jour. Il existe de nombreux modèles et protocoles de cohérence offrant différents niveaux de garanties/performances. Cependant, il n'existe pas de solution universelle. Offrir un mécanisme de maintien de la cohérence de données répliquées à grande échelle est donc toujours un problème ouvert pour de nombreuses applications.

5.1.6 Applications de la réplication

La réplication est largement utilisée pour répondre aux problèmes issus de la distribution : elle améliore les performances d'accès, assure la tolérance aux pannes et facilite le passage à l'échelle.

— **Amélioration des performances d'accès:** L'implication des réseaux dans les interactions distribuées introduit une différence entre les accès locaux et les accès distribués pouvant atteindre plusieurs ordres de grandeur. Les performances d'accès peuvent être améliorées par réplication des services sur des machines ayant des connexions réseau de bonne qualité. Cette technique est typiquement utilisée dans les systèmes de cache et dans les environnements à usagers mobiles. Dans le cas de ces derniers, la réplication est guidée par les déplacements des usagers et leur fournit un accès continu à leurs environnements habituels de travail.
— **Tolérance aux pannes:** La tolérance aux pannes est une des utilisations principales de la réplication. Les pannes de machines sont tolérées en copiant un service sur plusieurs machines, alors que les pannes de connexion sont gérées en copiant les services sur des machines accessibles par des chemins différents. Les pannes de connexion sont également gérées dans les environnements à usagers mobiles où la réplication permet le travail en mode déconnecté.
— **Passage à l'échelle:** La mise à l'échelle aggrave tous les problèmes issus de la distribution. Les environnements à grande échelle sont plus vulnérables aux pannes, subissent un trafic qui sature les canaux de communication et doivent faire face à un nombre de requêtes qui dépasse la capacité de traitement des services. La réplication de services permet de placer des copies partout dans les grandes infrastructures, de diminuer ainsi le trafic et de répartir la charge en aiguillonnant les clients vers les copies disposant de plus de ressources.

L'amélioration des performances d'accès, la tolérance aux pannes ou encore le passge à l'échelle sont nécessaires dans de nombreux domaines. Les protocoles de réplication et de cohérence trouvent ainsi des applications dans les systèmes de fichiers, les bases de données, les mémoires partagées réparties, les processus distribués, etc.

— **Systèmes de fichiers:** Les systèmes de fichiers sont les structures fondamentales pour l'organisation des informations. Ils utilisent la réplication pour assurer la pérennité des informations (tolérance aux pannes), pour garantir la performance d'accès, ainsi que pour mettre en place des solutions pour des usagers mobiles. Leur application principale est le stockage des données personnelles: documents écrits, documents multimédia, courriers, etc.
— **Bases de données:** Comme les systèmes de fichiers, les bases de données gèrent le stockage, l'organisation et la manipulation de données. Leurs principales préoccupations en ce qui concerne la réplication et la cohérence sont également la tolérance aux pannes, l'amélioration de performances et la prise en charge des usagers mobiles. Omniprésentes, elles ont leur place dans le secteur financier, dans les entreprises de service, dans les offres touristiques, etc.
— **Mémoires partagées réparties:** Fournissant l'abstraction d'une mémoire centralisée, le défi pour les mémoires partagées réparties est d'assurer la performance d'accès. Se focalisant sur la gestion de caches, les protocoles de réplication portent surtout sur les mécanismes permettant la réduction du nombre de messages de synchronisation. Typiquement utilisées pour les calculs scientifiques parallèles, les mémoires partagées réparties trouvent également leur place pour la gestion de caches web.
— **Processus distribués:** Les processus distribués ne considèrent pas les services du point de vue de leurs données, mais les représentent en termes de calculs composés de suites d'actions. Leur utilisation de la réplication et de la cohérence vise principalement la tolérance aux pannes. Les travaux existants se concentrent sur les mécanismes de gestion de groupes de processus et sur la gestion de la cohérence par diffusion ordonnée de messages. Les processus distribués sont un modèle de base des exécutions réparties et s'appliquent par conséquent à tout genre d'applications.

5.1.7 Caractéristiques des protocoles de réplication

Un protocole de réplication gère la cohérence entre les différentes copies d'un même objet tout en ayant pour objectif d'améliorer la fiabilité des données et/ou les performances (tant en écriture qu'en lecture) du système. Malheureusement, ces deux objectifs sont antagonistes (voir figure 5.1).

Pour obtenir une bonne fiabilité, il est nécessaire d'avoir une cohérence forte ce qui pénalise les performances. A l'inverse pour obtenir de bonnes performances, il est nécessaire de relâcher la cohérence, ce qui pénalise la fiabilité. Cela est d'autant plus vrai que l'on augmente le nombre de copies. En effet, dans le cas de la cohérence forte, un site accède toujours au dernier élément de la séquence globale des écritures. Ceci peut être réalisé par un protocole basé sur une diffusion atomique des valeurs ou par le verrouillage global de la donnée avant l'ajout d'un élément à la séquence. Ces méthodes se basent sur un ordre total sur les écritures. Par contre dans le cas de la cohérence faible, on n'assure que la valeur lue sur un site (la dernière valeur de la séquence locale) est bien la dernière de la séquence globale. Cela est généralement réalisé par la construction partielle sur les écritures. Il est donc nécessaire de faire des compromis. On peut distinguer deux types de cohérence pour un objet dupliqué:

1. Il y a cohérence forte entre les copies d'un même objet s'il y a ordre total sur les écritures faites sur ces copies.
2. Il y a cohérence faible entre les copies d'un même objet s'il y a ordre partiel sue les écritures faites sue ces copies.

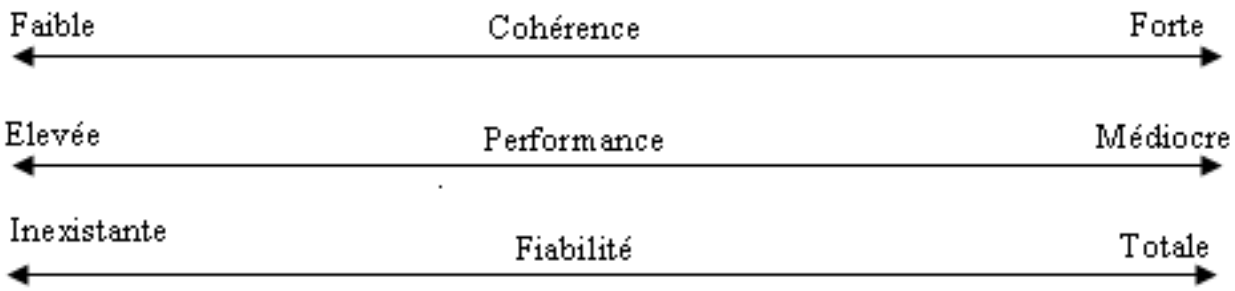


FIGURE 5.1 – Compromis Performance/fiabilité et répercussions sur la cohérence

De ce compromis performance/fiabilité résulte de nombreux protocoles de réplication mettant en œuvre une cohérence plus ou moins forte entre les copies que ce soit dans les SCG (Système de Communication de Groupe), les SGBDR (Les Systèmes de Gestion de Bases de Données Répartis) ou les MPR (Les Mémoires Partagés Réparties).

Gestion des fautes:

Certains protocoles de réplication peuvent supporter les fautes survenant sur les copies et d'autres non. Généralement, la tolérance aux fautes dans un protocole de réplication comporte deux aspects: la détection des fautes et la récupération. Bien souvent, pour les fautes par arrêt ou les fautes réseau, la détection se fait quand une copie ne répond plus. Pour les fautes par valeur ou byzantine, elle se fait par des algorithmes de vote. Dans ce cas, il est nécessaire de dupliquer les traitements. Une fois la faute détectée, la copie incriminée est supprimée du groupe des copies géré par le protocole jusqu'à ce qu'elle retrouve un comportement normal. Des traitements peuvent alors être nécessaires afin de retrouver une configuration normale pour le protocole. Avant de réintégrer une copie ayant à nouveau un comportement normal au groupe, il est nécessaire qu'elle rattrape son retard sur les autres copies.

Notion de copie:

Suivant les protocoles, la notion de copie peut être différente. Certains protocoles de réplication ne savent pas ce qu'est une copie. Ils s'en remettent à l'application pour créer et détruire les copies. De plus, un objet peut référencer d'autres objets. Suivant les cas, il est possible qu'il soit nécessaire de dupliquer ces références lors de la création d'une copie. Différents modèles de données sont également à considérer: orienté objet, relationnel, fichier, autre ou ouvert. Le terme « ouvert » sigifie que le protocole supporte différents modèles. Il est également nécessaire de décider combien de copies dans le système sont nécessaires?; où les mettre?; A quels moments en créer de nouvelles? Cette allocation des copies peut être dynamique ou statique.

Transparence à la réplication:

Le protocole de réplication peut offrir plus ou moins de transparence à l'application. Il y a transparence à la réplication si l'application n'a pas conscience du fait que les objets sont dupliqués. L'application manipule les objets dupliqués comme des objets classiques ; on parle d'objets logiques. S'il n'y a pas transparence à la réplication; l'application manipule explicitement chaque copie. On parle alors d'objets physiques.

5.1.8 Classification des travaux relatifs à la réplication

Le critères de classification que nous avons retenus pour la réplication sont les suivants:

— *Unité de réplication (Quoi?)*. Quelles sont les entités dupliquées? Quel est leur type?
— *Technique de copie (Comment?)*. Comment une copie est-elle crée? Quelles sont les informations dupliquées?
— *Moment de copie (Quand?)*. Quand une copie est-elle crée? Est-elle crée statiquement,avant l'exécution, ou dynamiquement, en cours d'exécution?
— *Placement de copie (Où?)*. Où une copie s'exécute-t-elle? Comment le placement répond-il aux objectifs de passage à l'échelle, de performances et de tolérance aux pannes?

Unité de réplication (Quoi?)

Un protocole de réplication est caractérisé par le type des copies qu'il gère, ainsi que par les critères de sélection des entités effectivement copiées. Le type des copies est généralement choisi en fonction des domaines d'application des protocoles, alors que les critères de sélection sont définis en fonction des objectifs de performances des protocoles. Ces deux points sont détaillés comme suit.
(i) Type de copie: Le type caractérise la structure et la sémantique d'une entité. La structure définit l'organisation interne de l'entité et joue un rôle important lors du processus de création de copie. Ce processus consiste à reproduire la structure et le contenu de l'entité.
La sémantique définit les opérations autorisées sur une entité. Elle définit des critères de correction qui sont à la base de la gestion de la cohérence entre copies. Par exemple, la sémantique des fichiers stipule que deux opérations d'écriture sur un même fichier sont conflictuelles mais que les opérations de création et de destruction de fichiers différents dans un répertoire ne le sont pas. La manière dont ces critères de correction sont définis et préservés est dédiée aux travaux relatifs à la cohérence. Les protocoles de réplication considèrent principalement quatre types d'entités : les fichiers, les zones mémoire, les bases de données et les objets.

— ***fichiers:*** Dans le cas des fichiers, les techniques de copie se basent sur la structure séquentielle des fichiers et sur leur organisation hiérarchique en répertoires. La cohérence de copies de fichiers est maintenue par des synchronisations des accès en lecture et en écriture. La réplication garantit la tolérance aux pannes dans les systèmes. Elle assure le travail en mode déconnecté pour les usagers mobiles dans de nombreux projets. Elle permet l'accès rapide aux fichiers dans les travaux s'intéressant aux environnements à grande échelle .

— ***Zones mémoire:*** La réplication de la mémoire est appliquée dans les travaux sur les mémoires partagées réparties avec l'objectif de fournir l'abstraction d'un environnement d'exécution centralisé. Comme dans le cas des fichiers, la cohérence est gérée par rapport aux accès en lecture et en écriture, et la copie est définie en fonction des structures hiérarchiques de blocs et de pages.
— ***Bases de données:*** Comme les mémoires et les fichiers, les bases de données représentent un support essentiel pour le stockage de données et utilisent la réplication pour la tolérances aux pannes, les performances d'accès, le travail des usagers mobiles en mode déconnecté, etc. Toutefois, la possibilité de définir différents types de données impose l'utilisation de techniques de copie plus complexes qui prennent en compte la spécificité des données. La gestion de la cohérence est également influencée et considère, au moyen des transactions, des opérations plus évoluées que les lectures et les écritures de données.
— ***Objets:*** Étant une abstraction qui permet la modélisation de tout type d'entité, les objets sont largement utilisés et dupliqués pour les besoins de tolérance aux pannes, de performance d'accès et de passage à l'échelle.

(ii) Sélection des entités à copier: Les protocoles de réplication appliquent des critères de sélection pour décider si une entité doit être effectivement copiée ou non. Ces critères sont principalement de trois types : ils sont définis en fonction des types des entités, portent sur les modèles d'accès à ces entités ou alors sont des critères applicatifs.

— ***Critères de type:*** Les critères de type sont utilisés pour distinguer les entités de type "duplicable" d'un système et pour indiquer que ces entités sont systématiquement dupliqués.
— ***Critères d'accès:*** Les critères d'accès limitent les coûts de synchronisation en restreignant l'ensemble des entités à copier. Ils copient les entités uniquement si celles-ci sont effectivement utilisées et sont appliqués dans les protocoles de gestion de cache.
— ***Critères applicatifs:*** Alors que les critères de type et les critères d'accès sont définis au niveau des protocoles de réplication, les critères applicatifs sont définis par les applications qui utilisent ces protocoles. En prenant en compte les spécificités applicatives, ces critères visent à réduire le coût de synchronisation de copies ainsi que l'espace de stockage nécessaire à celles-ci. Ils ont typiquement été mis en place sous forme de préférences utilisateur dans les environnements à usagers mobiles et sous forme d'abonnements dans les réseaux actifs diffusant du contenu à la demande.

Technique de copie (Comment?):

Le processus de création de copie dépend de la structure et de l'état de l'entité à dupliquer. La structure de l'entité peut être indivisible ou composée, alors que l'état peut être constitué de données, de code et d'un éventuel état d'exécution.
(i) Structure de copie: Dans le cas où l'entité à copier a une structure indivisible, la copie reproduit cette structure et l'initialise avec l'état capturé de l'entité. La technique de copie dépend donc de l'état de l'entité.

Dans le cas où l'entité à copier a une structure composée, la copie peut être faite à différents niveaux de *granularité*. En effet, elle peut considérer la structure dans sa totalité ou alors travailler sur les structures qui font partie de cette dernière. La granularité de copie influence les performances des protocoles. Une grande granularité permet la copie groupée de plusieurs entités et répond aux besoins de localité d'accès. Toutefois, elle interdit les accès concurrents aux entités appartenant à un tel groupe. Une granularité fine atténue le problème de partage mais induit d'importants coûts de synchronisation.

(ii) état de copie: La création de copie est basée sur la capture et la restauration de l'état de l'entité à dupliquer. Dans le cas d'entités données, la capture et la restauration travaillent sur les valeurs de ces données. Dans le cas d'entités objets, la capture et la restauration considèrent les

données internes des objets, ainsi que leur état d'exécution. Toutefois, la complexité de capture de l'état d'exécution fait que la plupart des techniques ne considèrent que l'état "passif" des objets.

Moment de copie (Quand?):

La création de copies peut être faite de manière statique ou dynamique. La création statique correspond à la définition de copies d'une entité avant l'exécution de celle-ci, alors que la création dynamique permet la création de copies en cours d'exécution. La création statique de copies a l'avantage de fixer le nombre, les identités et les placements des copies. La connaissance de ces informations permet la mise en place de protocoles de cohérence adaptés à la configuration spécifique de l'ensemble de copies. Elle peut être utilisée, par exemple, pour la détection de pannes de copies comme dans le cas de NFS(*Network File System*).

La création dynamique de copies permet l'adaptation de l'ensemble de copies aux variations de l'environnement d'exécution. Les facteurs qui déclenchent de telles créations dynamiques sont typiquement les problèmes de connexion, les problèmes de surcharge ou encore les nombres d'accès consécutifs aux copies. Les nouvelles copies répondent à ces problèmes en assurant respectivement la disponibilité lors de partitions, la répartition de charge et la localité d'accès.

Placement de copie (Où?):

Le placement des copies a un impact direct sur le fonctionnement d'un système dupliqué. Les schémas de placement sont définis en fonction des objectifs de performance d'accès, de tolérance aux fautes ou encore de passage à l'échelle.

Pour une amélioration des performances d'accès, le placement doit prendre en compte les caractéristiques des environnements d'exécution des utilisateurs.

5.1.9 Problème du déterminisme de tâches dupliquées

Le problème du déterminisme de tâches dupliquées est comment garantir qu'aucun effet de bord non désiré autre qu'une faute ne mène à une incohérence entre les copies d'une tâche?

Si les tâches dupliquées ne sont pas déterministes, il n'est pas possible de garantir qu'après le recouvrement d'une erreur, les résultats calculés par les copies d'une tâche dupliquée placées sur des calculateurs non défaillants sont corrects. Le déterminisme des tâches dupliquées est donc une condition nécessaire pour mesurer les performances d'une technique de réplication. La définition exacte du déterminisme de tâches dupliquées est liée à la technique de réplication et au modèle de tâches. De manière intuitive, les tâches dupliquées sont déterministes si pour toute exécution des copies d'une tâche placées sur des calculateurs non défaillants, les messages de calcul émis par chaque copie de cette tâche sont identiques et sont cohérents avec les modifications effectuées.

Cette définition suppose que les mêmes messages de calcul sont livrés aux copies de chaque tâche dupliquée et qu'initialement les copies de chaque variable persistante sont identiques. Les principales sources d'indéterminisme potentiel sont les suivantes:

— **Constructions logicielles indéterministes:** certains langages ont des constructions logicielles qui changent l'ordre d'exécution de portions de code(par exemple la clause select du langage Ada);

— **Informations locales aux calculateurs:** l'architecture matérielle ou le système d'exploitation d'un calculateur peuvent fournir des informations locales qui diffèrent d'un calculateur à l'autre, même si elles sont consultées au même instant. Parmi ces informations, on peut distinguer la date locale (puisque sans horloge commune, il n'est pas possible de synchroniser parfaitement la date locale de plusieurs calculateurs), ou un générateur de nombres aléatoires.

— **Ordre différent de livraison des messages de calcul aux copies des tâches:** si deux messages de calcul sont diffusés aux copies d'une tâche, alors ces messages peuvent être livrés dans un ordre différent selon les calculateurs, et donc être consommés dans un ordre différent

selon les copies de la tâche. Ils peuvent également provoquer des décisions d'ordonnancement différentes pour les copies des deux tâches.

— **Décisions d'ordonnancement des copies des tâches:** les décisions d'ordonnancement pour les copies des tâches sont identiques si, pour toute exécution des copies de tâches $T1$ et $T2$, sur chaque calculateur où est placée une copie de $T1$ et de $T2$:
   1. soit l'exécution de la copie de $T1$ (respectivement $T2$) est terminée avant que ne commence celle de $T2$ (respectivement $T1$) ;
   2. soit l'exécution de la copie de $T1$ (respectivement $T2$) est suspendue au profit de celle de la copie de $T2$ (respectivement $T1$) à la même position dans le flot d'exécution de la copie de $T1$.

Des décisions d'ordonnancement différentes pour les copies des tâches provoquent donc l'exécution de séquences d'instructions différentes selon les calculateurs. Un algorithme d'ordonnancement peut prendre des décisions d'ordonnancement qui sont différentes pour les copies des tâches si l'ordre d'exécution des copies placées sur un calculateur est calculé en fonction d'informations variant d'un calculateur à l'autre (par exemple le temps d'exécution d'une copie de tâche, ou l'ordre de livraison de messages de calcul diffusés aux copies de plusieurs tâches). Il peut également prendre des décisions d'ordonnancement différentes si deux calculateurs qui exécutent deux copies d'une tâche n'exécutent pas les mêmes ensembles de tâches.

5.2 Protocoles de réplications (Passif, Actif, Semi-actif)

Dans les systèmes distribués, trois modes principaux de réplication sont envisageables : la réplication active, semi-active ou passive. Le choix entre ces différents mécanismes se fait selon les hypothèses de fautes et le domaine d'application [Ben05].

5.2.1 Réplication active (N-modules replication)

dans ce protocole de réplication, chaque réplique joue le même rôle : chacune reçoit la requête émise par le client, la traite, met à jour son état interne puis retourne sa réponse au client émetteur. Dans ce cas, le protocole de réplication n'est pas centralisé puisque chaque réplique fournit les réponses. Par ailleurs, et comme les requêtes sont envoyées à toutes les répliques, la défaillance de l'une d'entre elles est transparente du point de vue du client. En effet, si on prend comme hypothèses que $(H1)$ toutes les répliques sont déterministes, que $(H2)$ le canal de communication est non-défaillant, que $(H3)$ les requêtes arrivent et sont traitées par les serveurs dans le même ordre et que $(H4)$ ces derniers ne défaillent que sur arrêt, la première réponse qui arrive au client sera considérée comme correcte. Ceci masquerait, en cas de retour de réponse, la défaillance probable d'au plus $N-1$ serveurs, N étant le nombre total des répliques. Par ailleurs, si l'hypothèse $(H4)$ est relaxée (i.e. un serveur ne peut retourner une valeur erronée), un vote sur les réponses retournées sera réalisé et c'est la valeur issue du vote qui sera délivrée au client. Dans ce cas, on aura besoin de $2N+1$ répliques pour pouvoir tolérer N erreurs par valeurs simultanées.

5.2.2 Réplication passive (primary-backup replication)

à l'inverse de la réplication active, ce protocole est centralisé au niveau du serveur primaire ; les autres répliques appelées aussi secondaires (backups) ne reçoivent pas la requête du client. En effet, dans cette stratégie, un client envoie une requête au primaire uniquement, celui-ci l'exécute, met à jour l'état des secondaires à partir de son propre état, puis retourne la réponse au client. Si le primaire défaille (sur arrêt), un serveur parmi les répliques secondaires prend la relève. Par ailleurs, en utilisant la réplication passive, nous n'avons plus besoin de l'hypothèse de l'atomicité $(H3)$, puisqu'elle est intrinsèque d'une part à la linéarisation des messages reçus et envoyés par le primaire, et d'autre part, à l'attente de l'acquittement envoyé par les secondaires et attendu par

le serveur primaire. L'hypothèse du déterminisme $(H1)$ n'est pas non plus nécessaire, puisque seul le serveur primaire traite la requête, ce qui distingue cette approche de la réplication active en termes d'utilisation des ressources. Ainsi, un tel mécanisme de réplication est le plus rapide pour le traitement des requêtes en l'absence de fautes. En revanche, cette approche est plus lente pour retourner une réponse dans le cas d'une défaillance du primaire. En plus, elle est plus difficile à mettre en œuvre puisque l'application qui l'utilise doit fournir des mécanismes de mise à jour de l'état (capture et envoi). Cette approche est très utile pour les applications sans changement d'état comme dans le cas de la consultation et non pas de mise à jour des bases de données.

5.2.3 Réplication semi-active (leader-follower replication)

à mi-chemin entre la réplication active et la réplication passive, cette approche définit un meneur (leader) et plusieurs suiveurs (followers) qui traitent les requêtes envoyées par le client (comme dans le cas de la réplication active). En revanche, seul le leader retourne une réponse (comme dans le cas de la réplication passive). Au cas où le leader défaille par arrêt, une réplique parmi l'ensemble des secondaires prend la relève. Cette approche est caractérisée par la rapidité du recouvrement et par la non nécessité de la mise à jour de l'état puisque toute les requêtes sont exécutées parallèlement. En revanche, le déterminisme de toutes les répliques est nécessaire puisque chacune d'entre elles gère la mise à jour de son état interne.

5.3 Résumé du chapitre

Dans les systèmes parallèles où la durée du calcul est très importante, toutes les ressources disponibles doivent être utilisées pour le calcul lui-même.

La tolérance aux fautes par réplication où on utilise une redondance suffisante pour masquer les conséquences d'une erreur. Les processus sont dupliqués, et dans le cas d'une panne, le processus défaillant est remplacé par une copie. L'avantage de cette technique est le gain de temps par rapport aux techniques de recouvrement, l'inconvénient est l'utilisation d'un nombre élevé de ressources (noeuds), et la difficulté du maintien des copies dans le même état que le processus original.

Il y a principalement deux raisons pour répliquer les données : améliorer la fiabilité d'un système distribué et l'amélioration des performances. La réplication introduit un problème de cohérence: chaque fois qu'une réplique est mise à jour, cette réplique devient différente des autres. Pour garder les répliques cohérentes, nous devons propager les mises à jour dans de telle manière à ce que les incohérences temporaires ne soient pas remarquées. Malheureusement, cela peut dégrader gravement les performances, en particulier dans les systèmes distribués à grande échelle.

5.4 Exercices

Exercice 5.1 L'accès aux objets partagés Java peut être sérialisé en déclarant ses méthodes comme étant synchronisées (Synchronized). Est-ce suffisant pour garantir la sérialisation lorsqu'un tel objet est répliqué ? ■

Exercice 5.2 Pour que la réplication active fonctionne en général, il est nécessaire que toutes les opérations soient effectuées dans le même ordre à chaque réplique. Cette commande est-elle toujours nécessaire ? ■

6. Tolérance aux pannes dans les SR

6.1 Définition, classification et critères de tolérance aux pannes

Le terme "Tolérance de panne" est très large . Il regroupe les pannes de communication, les bugs informatiques , les pannes de machines , les défaillances de périphériques de stockage ou tout affaiblissement d'un support de données. Les degrés de tolérance varient suivant les contraintes que doit supporter le système à déployer. Un système tolérant aux pannes doit pouvoir continuer à fonctionner éventuellement sous une forme dégradée , même après l'apparition de pannes ou d'états indésirables.

Une panne peut être déclarée alors qu'une ressource est utilisée de façon trop importante. Le temps d'accès devient trop long et le système la considère en panne. La tolérance au panne et l'adaptabilité d'un système distribué dépendent l'une de l'autre. Un systèmes distribué tolérant aux pannes doit être capable de:

- l'ajout et la suppression dynamique des ressources.
- le traitement de la panne d'un ou plusieurs nœuds sans avoir à redémarrer globalement l'application.
- la reprise sur n'importe quel autre type de nœud disponible d'un ou plusieurs nœuds défaillants.
- la minimisation, dans le système, du nombre de composants fiables durant toute l'exécution de l'application.

Définition 6.1.1 Menace: Une menace est une cause potentielle d'incident, qui peut résulter en un dommage au système ou à l'organisation (définition selon la norme de sécurité des systèmes d'information ISO/CEI 27000).

Définition 6.1.2 Faute: Défaut physique du matériel ou du logiciel.Une faute ou panne est caractérisée par sa nature, son origine et son étendue temporelle

Définition 6.1.3 Erreur Une valeur incorrecte dans le système. Une erreur est la conséquence d'une faute, le service ne correspond pas en valeur à celui spécifié, le service n'est pas délivré dans l'intervalle de temps spécifié.

Définition 6.1.4 Défaillance / Panne Une défaillance définie l'incapacité d'un élément du système à assurer le service spécifié par l'utilisateur. La défaillance est caractérisée par son domaine, sa perception par les utilisateurs et ses conséquences sur l'environnement

6.2 Détecteurs de défaillance de Chandra et Toueg et Méthodes

Les détecteurs de défaillances ont été introduit par Chandra et Toueg en 1996 et ont été l'objet d'une recherche active depuis cette date. On présentera les détecteurs de défaillances, les notions de réduction et de plus faible détecteur de défaillances pour résoudre un problème en présence de panne.

6.2.1 Détecteur de défaillance

Un détecteur de pannes est un service réparti composé de détecteurs locaux attachés à chaque processus (ou site) . Un détecteur fournit sur demande la liste des processus qu'il soupçonne d'être défaillants. Les divers détecteurs locaux coopèrent entre eux pour établir cette liste [Kra].

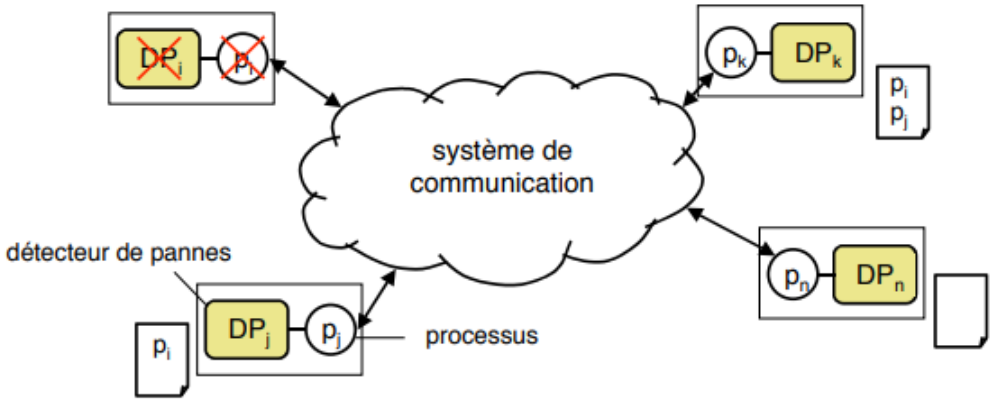


FIGURE 6.1 – Détecteur de pannes

6.2.2 Classes des détecteurs de défaillances

complétude	justesse			
	forte	faible	finalement forte	finalement faible
forte	parfait P	fort S	finalement parfait ◇ P	finalement fort ◇ S
faible	quasiment parfait Q	faible W	finalement quasiment parfait ◇ Q	finalement faible ◇ W

FIGURE 6.2 – Classes des détecteurs de défaillances

Hypothèse : pannes franches sans reprise, communication fiable asynchrone.

— **Complétude (completeness):** Le détecteur doit détecter les processus fautifs. Plus précisément :

  - **Complétude forte :** tout processus fautif finit par être soupçonné par tout processus correct
  - **Complétude faible :** tout processus fautif finit par être soupçonné par un processus correct
- **Exactitude (accuracy):** Le détecteur ne doit pas déclarer fautif un processus correct. Plus précisément :
  - **Exactitude forte :** aucun processus correct n'est jamais soupçonné par un autre processus correct
  - **Exactitude faible :** il existe un processus correct qui n'est jamais soupçonné par un autre processus correct .
- On peut encore affaiblir ces propriétés en considérant le temps
  - **Exactitude finalement forte :** au bout d'un certain temps, aucun processus correct n'est plus jamais soupçonné par un processus correct
  - **Exactitude finalement faible :** au bout d'un certain temps, il existe un processus correct qui n'est plus jamais soupçonné par un autre processus correct

  NB : "finalement" traduit eventually ; on peut aussi dire "inévitablement"

A priori, 8 classes de détecteurs. En réalité, complétude forte et faible sont équivalentes. On sait réaliser la complétude forte si on dispose de la complétude faible . Donc 4 classes distinctes : $P, \Diamond P, S, \Diamond S$.
La flèche signifie : "implique" (si on assure P, on assure S, etc.

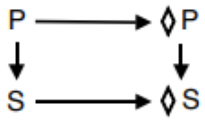


FIGURE 6.3 –

Complétude faible vs complétude forte

Hypothèse : pannes franches, communication fiable asynchrone

Supposons que l'on dispose d'un détecteur assurant la complétude faible.

Ce détecteur garantit que tout processus fautif finit par être soupçonné par un processus correct. Donc au bout d'un temps fini T, un processus correct, soit p, possède la liste de tous les processus fautifs. Si tout processus diffuse périodiquement sa liste (par diffusion fiable), tous les processus fautifs seront soupçonnés au bout d'un temps fini par tout processus correct (d'après les propriétés de la diffusion fiable).

Donc complétude faible implique complétude forte. Comme on a aussi l'implication inverse, les deux formes de complétude sont équivalentes.

Le détecteur de pannes parfait

Qualités

Un détecteur parfait (de classe P) possède les qualités

de complétude forte

d'exactitude forte

Capacités

Un détecteur parfait permet de résoudre le consensus en asynchrone avec pannes franches .

6.3 Les techniques de la tolérance aux pannes

La tolérance aux pannes est toujours traitée par l'emploi d'une redondance des calculs (traitements multiples) ou des moyens de stockage (réplication de composants) ou bien informationnelle (redondance de données, codes, signatures). Dans un tel système à base de processus communicants, les techniques de tolérance aux pannes peuvent être séparées en deux classes : les techniques basées sur la réplication et les techniques basées sur une mémoire stable.

6.3.1 Tolérance aux pannes par réplication

La tolérance aux pannes par réplication est basée sur la création de copies multiples des composants sur des processeurs différents. Cette approche rend le traitement des pannes possible en les masquant. Trois stratégies principales pour réaliser la réplication sont proposées dans la littérature : les réplications actives, passives et semi-actives. Ces différentes stratégies visent à garantir une cohérence forte entre les copies d'un composant dupliqué [Sai12].

réplication active :

elle est définie par la symétrie du comportement des copies d'un composant dupliqué où chaque copie joue un rôle identique à celui des autres. C'est la plus utilisée dans les applications courantes comme les systèmes d'information.

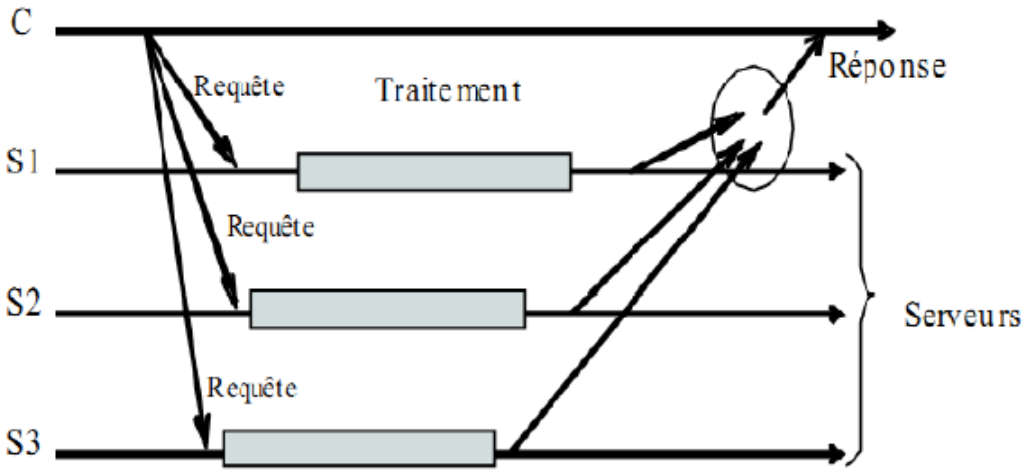


FIGURE 6.4 – Protocole de réplication active

Tolérance aux pannes:

Dans le protocole de réplication active, quand une copie devient défaillante, aucun mécanisme supplémentaire n'est à mettre en oeuvre. La défaillance d'une copie est masquée par le comportement des copies non défaillantes. Ainsi, la tolérance aux fautes est réalisée par le mécanisme de masquage d'erreur. L'exécution des requêtes doit être déterministe, sinon des mécanismes de vote sont nécessaire, après la collecte des réponses de toutes les différentes copies, pour décider de la réponse à restituer au client.

réplication passive :

contrairement à la réplication active, la réplication passive (passive replication ou primary-backups replication) est asymétrique. Elle distingue deux comportements pour les copies d'un composant dupliqué : la copie primaire (primary copy) et les copies secondaires (backups). La copie primaire est la seule à effectuer tous les traitements.

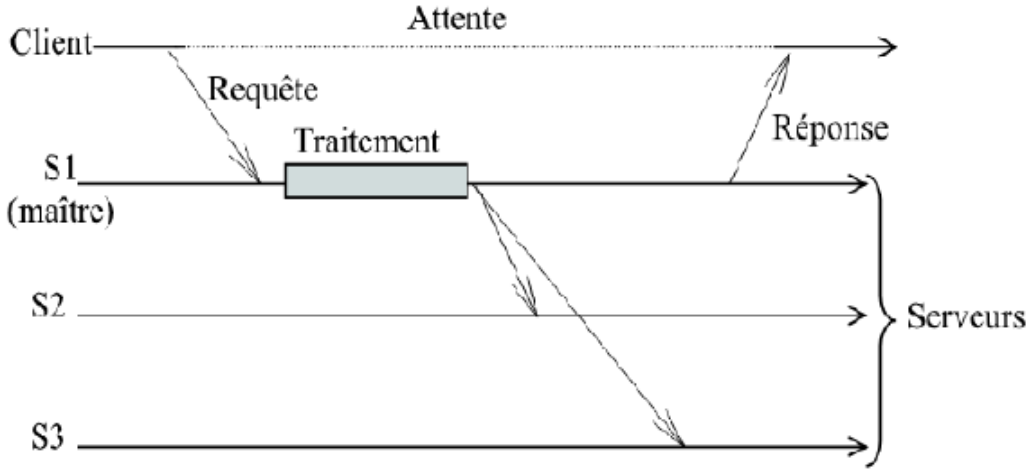


FIGURE 6.5 – Protocole de réplication passive

Les copies secondaires, oisives, surveillent la copie primaire. En cas de défaillance de la copie primaire, une des copies secondaires devient la nouvelle copie primaire. Elle est utilisée dans des applications critiques, temps réel.

Tolérance aux pannes:

La tolérance aux fautes est réalisée par détection et compensation d'erreur. >Quand une copie secondaire passe dans un état de défaillance, aucun traitement particulier n'est nécessaire. Le seul effet de cette défaillance est la diminution du degré de réplication (une réplique au moins). Par contre, quand une copie primaire devient défaillante, un protocole d'élection sera déclenché pour désigner la nouvelle copie primaire parmi les copies secondaires. Si la copie primaire fait défaut lors d'une invocation par un client, alors celui-ci n'obtient aucune réponse à sa requête. Il doit ré-émettre sa requête en l'adressant à la nouvelle copie primaire.

réplication semi-active :

la réplication semi-active (semi-active replication ou leader followers replication) se situe à mi-chemin entre la réplication active et la réplication passive. Comme cette dernière, la réplication semi-active est une stratégie asymétrique. Contrairement à la réplication passive, les copies secondaires ne sont pas oisives.

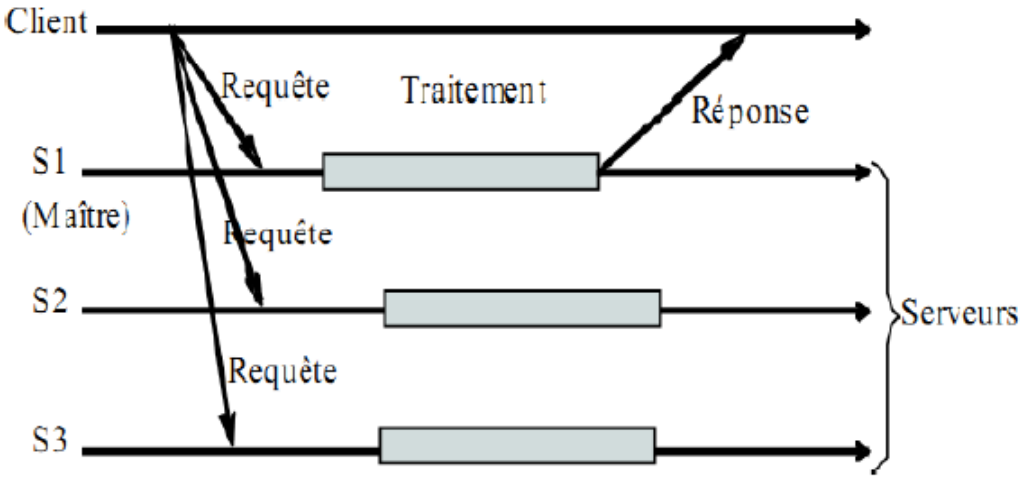


FIGURE 6.6 – Protocole de réplication semi active

Tolérance aux pannes:

Dans la réplication semi active, quand une réplique suiveuse fait défaut aucun traitement particulier n'est nécessaire. Son seul effet est de diminuer le degré de réplication. Puisque toutes les copies reçoivent la requête, le client n'a pas besoin de ré-émettre sa requête lorsque la copie maître devient défaillante. La nouvelle copie maître envoie automatiquement la réponse au client.

	Approche		
Critère de comparaison	**Redondance active**	**Redondance passive**	**Redondance semi-active**
Surcoût	un surcoût élevé	un surcoût moins élevé	le surcoût dépend du niveau de la réplication active par rapport à la réplication passive
Détection de défaillance	pas besoin de détecter les défaillances	mécanisme spécial de détection de défaillances	mécanisme spécial, coûteux et souvent compliqué
Traitement de défaillances (Temps de réponse)	un temps de réponse prévisible, et généralement rapide dans des architectures offrant un taux élevé de parallélisme	meilleur temps de réponse en absence de défaillances. La défaillance de la réplique primaire peut de manière significative augmenter le temps de réponse	le temps de réponse dépend du niveau de la réplication active par rapport à la réplication passive
Reprise après défaillance	immédiate	non immédiate	non immédiate

FIGURE 6.7 – Comparaison entre les trois approches de redondance

La figure 6.7 donne une indication comparative des différentes approches de tolérance aux

fautes basées sur la redondance active, passive ou hybride des composants logiciels d'un algorithme.

Une propriété intéressante de la redondance active se situe dans le fait qu'une faute n'augmente pas la latence du système temps réel, ce qui n'est pas le cas dans la redondance passive, où la faute de la réplique primaire peut de manière significative augmenter la latence du système.

Cependant, la redondance passive présente l'avantage de réduire la surcharge sur les processeurs et sur le réseau de communication, ce qui permet une meilleure exploitation des ressources matérielles offertes par l'architecture.

6.3.2 Tolérance aux pannes par mémoire stable

L'existence d'une mémoire stable permet la réalisation de la tolérance aux pannes par une détection de défaillance suivie d'un recouvrement d'erreur. Mémoire stable : La mémoire stable n'est qu'une abstraction. Elle peut être définie comme un support persistant de stockage, dont le rôle principal est d'assurer une accessibilité et une protection aux données contre les pannes pouvant affecter le système. Ainsi, suite à une panne, un état correct ayant été stocké antérieurement à cette panne sur la mémoire stable reste accessible ; cela permet un retour du système à un état antérieur [Bou07].

Un support de stockage est vu comme une mémoire stable si et seulement si les trois conditions suivantes sont vérifiées :

1. Accessibilité : il existe à tout moment de l'exécution un chemin permettant d'accéder aux données sauvegardées même en présence des pannes.
2. Protection : les pannes affectant le système ou l'application n'altèrent pas les données.
3. Atomicité : les mises à jour des données sur le support de stockage se font de manière atomique.

Une fois qu'une panne est détectée, un mécanisme de recouvrement doit être mis en place pour traiter la panne identifiée. Il existe deux techniques de recouvrement:

La reprise : c'est la technique générale qui consiste à retourner vers un état antérieur dont on sait qu'il est correct. Cette technique nécessite donc la sauvegarde régulière de l'état du système ou de l'application.

La poursuite : c'est une technique spécifique qui consiste à reconstituer un état correct, sans retour en arrière. La reconstitution n'est souvent que partielle, d'où un service dégradé.

■ Exemple 6.1 ■

Communication par paquets : comment réagir à la perte d'un paquet ?

Solution 1 : Reprise
L'émetteur conserve une copie des paquets qu'il a envoyés, jusqu'à la réception de l'acquittement. En cas de perte (par délai de garde), le récepteur demande à l'émetteur de renvoyer le paquet manquant.

Solution 2 : Poursuite
Pour un type particulier d'applications, l'émetteur peut constituer (totalement ou partiellement) le contenu d'un paquet manquant en utilisant les paquets voisins (suivants, précédents), le récepteur peut alors poursuivre sans demander à l'émetteur.

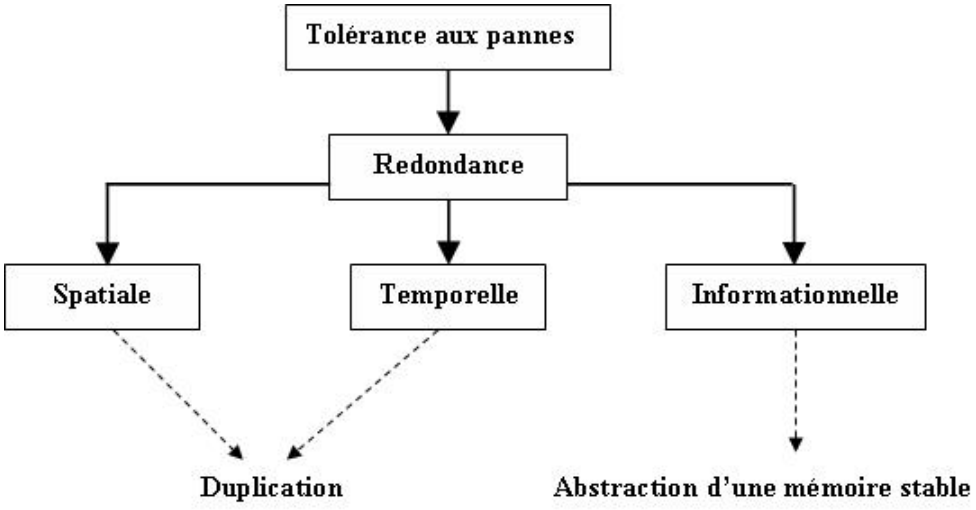


FIGURE 6.8 – Les techniques de tolérance aux pannes dans les systèmes distribués

6.4 Tolérance aux pannes par points de reprises

La tolérance aux pannes par points de reprise (checkpointing) utilise le fait qu'un processus en cours d'exécution est capable à tout moment de prendre un enregistrement complet de son état. Cet enregistrement sera utilisé en cas de panne comme point de reprise, ce qui évite au processus de répartir depuis le début de son exécution. Cet enregistrement doit correspondre à un état cohérent [1], c'est à dire un état par lequel passe un processus lors de l'exécution normale. Si cette notion de cohérence paraît triviale lorsque l'on considère un seul processus qui enregistre régulièrement son état, le problème devient complexe lorsqu'on veut obtenir un état cohérent d'un ensemble de processus communiquants. C'est principalement cette problématique que tentent d'adresser les protocoles de tolérances aux pannes par points de reprise : assurer la cohérence de l'ensemble des états de chaque processus.

6.4.1 Point de reprise

Un point de reprise est l'image de l'état local d'un processus. Il regroupe les informations suivantes:

le contexte mémoire : Il s'agit de l'espace d'adressage (virtuel) utilisé par le processus et des différents segments qui le compose. Dans un système de type Unix, le contexte mémoire regroupe au minimum un segment pour le code, un segment pour la pile et un segment pour les données et l'environnement d'exécution.

le contexte processeur : Il s'agit des différents registres du processeur. Par nature, les procédures de gestion du contexte processeur sont dépendantes de l'architecture et doivent être réécrite lors d'un portage.

le contexte système : Ce sont toutes les données qui caractérisent le processus au sein du système d'exploitation. De façon non exhaustive, nous pouvons citer les fichiers ouverts, les communications réseaux, les signaux ... Dans les systèmes à points de reprise orientés pour le support d'application de calcul parallèle, il est souvent supposé que seuls les fichiers standards (i.e., entrée, sortie, erreur) sont utilisés ; le reste du contexte système est négligé. La sauvegarde de l'état local du processus

1. nous considérerons toujours qu'un point de reprise pris par un processus unique représente un état cohérent de celui-ci.

peut être optimisée soit en réduisant le volume de données à sauvegarder, soit en évitant de bloquer le processus lors de l'établissement du point de reprise.

6.4.2 Cohérence d'un état global

Considérons un Considérons un système de n processus communiquants : nous allons montrer qu'un ensemble d'enregistrements locaux, représentant chacun un état cohérent d'un processus ne représente pas toujours un état cohérent de l'ensemble du système.

Définition 6.4.1 **Un état global** d'un système réparti est un ensemble d'états locaux dans lequel on trouve un enregistrement d'état local par processus.

Dans la figure 6.9, nous montrons trois processus ayant chacun pris un enregistrement local de leur état ; pourtant, l'ensemble de ces trois points ne représente pas un état global cohérent. En effet, **un état cohérent** est un état par lequel peut passer une exécution normale (sans contradiction).

Or, on voit ici que si chaque processus repart depuis son point de reprise, le message $m3$ à déja été reçu par le processus R, alors qu'il n'a pas encore été envoyé par Q On parle alors de message orphelin.

Définition 6.4.2 **Un message orphelin** est un message qui a été envoyé après un enregistrement d'état local et reçu avant par un autre enregistrement d'état local.

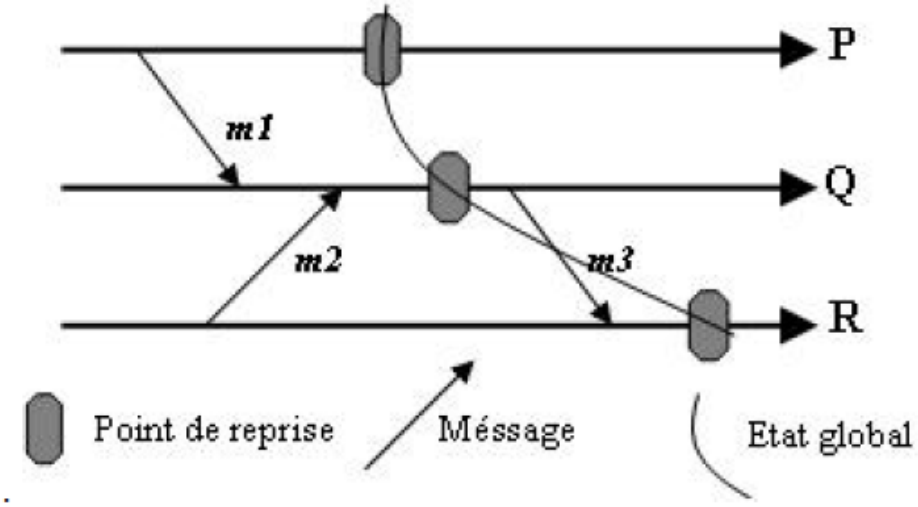


FIGURE 6.9 – Etat global incohérent

Cette notion de cohérence à été formalisée par Chandy et Lamport.

Définition 6.4.3 Un état global est **cohérent** s' il n'existe pas de message orphelin par rapport à cet état (**cohérence simple**).

En d'autres termes, si un point de reprise sur le processus P a enregistré la réception d'un message provenant du processus Q, alors il faut que le point de reprise sur Q appartenant au même état global ait enregistré l'émission de ce message. On note qu'un message peut avoir été envoyé mais non reçu (message $m3$ sur la figure 6.10) ; l'état global reste cohérent, au sens de ce message qui sera perdu en cas de reprise du système est un message en transit.

Définition 6.4.4 Un message en **transit** est un message qui a été envoyé avant un enregistrement d'état local et reçu après un enregistrement d'état local.

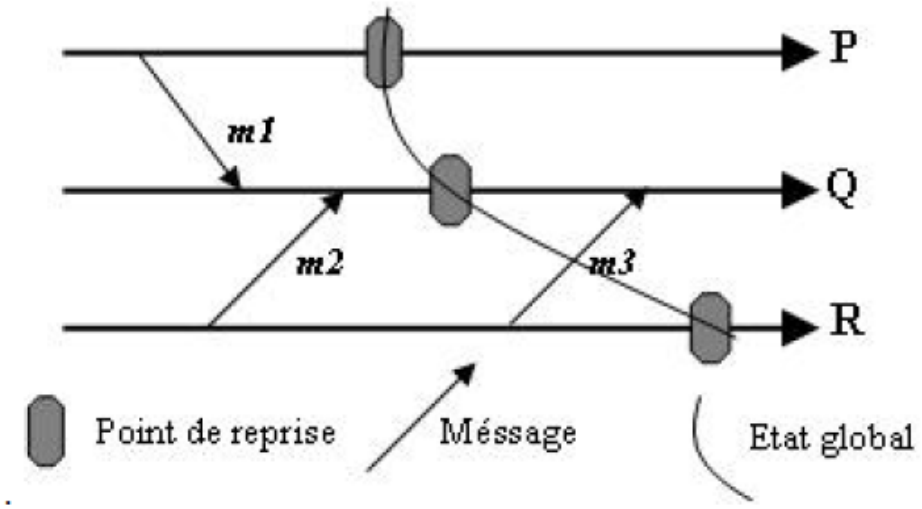


FIGURE 6.10 – Etat global cohérent

■ **Exemple 6.2** ■

Définition 6.4.5 La ligne de recouvrement est l'état global cohérent le plus récent.

Cette ligne de recouvrement doit toujours être assez récente, de façon à minimiser le retour arrière en cas de reprise, c'est à dire le nombre de points de reprise pris entre le dernier pris et celui utilisé pour la reprise. Nous verrons cependant que certains protocoles ne créent pas de la ligne de recouvrement durant l'exécution, mais recherchent cette ligne dans l'ensemble des états globaux formés, après qu'une panne soit survenue dans le systeme. Ces protocoles sont fortement sujets a ce qu'on appelle **l'effet domino**.

Définition 6.4.6 L'effet domino : l'état initial est le seul état global cohérent de système

Considérons la figure 6.11 : lors de la panne du processus P, le système va tenter de reprendre depuis l'état global le plus récent, c'est à dire celui formé par les points de reprise C_P^3, C_Q^3 et C_R^3 . à cause du message orphelin m6, cet état n'est pas cohérent : on va donc faire un retour arrière sur R, jusqu'au point C_R^2.

Mais l'état global ainsi formé n'est pas toujours cohérent. De la même manière, le système va devoir reculer jusqu'à la ligne de recouvrement (formée de C_P^1, C_Q^1, et C_R^1) qui représente l'état initial et donc perdre une grande quantité de travail.

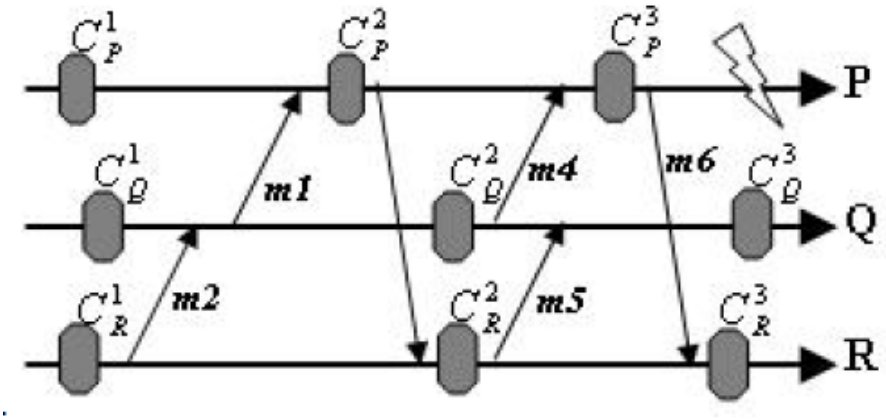


FIGURE 6.11 – Effet domino

Caractérisation de la cohérence

Nous donnons ici certaines conditions pour qu'un état global soit cohérent. On distingue deux manières de caractériser la cohérence. La première est basée sur la relation(happendbefore) de Lamport que l'on définit ainsi:

Soient A et B deux événements. A précède directement B ssi l 'une des conditions suivantes est vérifiée :

A et B se produisent sur le même site et A est antérieur à B sur ce site.

A est l'envoi d'un message m sur un site et B est la réception de m sur un autre site.

Cette relation de causalité entre les évènements permet de définir une condition nécessaire pour qu'un état global soit cohérent.

Propriété : Si deux points de reprise appartiennent tous les deux à un état global cohérent alors il n'existe pas de dépendance causale entre eux.

En effet, s' il existe une dépendance causale entre deux points de reprise du même état global, alors il existe dans cet état global deux point de reprise C_P sur le processus P et C_Q sur le processus Q qui sont en dépendance causale directe, c'est à dire qu'un message a été envoyé après la prise de C_P et reçu avant la prise de C_Q. Ce message est donc un message orphelin, et l'état global considéré n'est pas cohérent.

De cette manière, on peut vérifier que chaque communication directe (point-à-point) n'entraîne pas une dépendance causale entre un point de reprise de l'émetteur et un point de reprise du récepteur, et prendre les mesures nécessaires si c'est le cas, comme par exemple prendre un point de reprise avant l'émission ou la réception.

Cependant, cette règle n'est utilisable que si on considère les communications directes entre processus : il faut qu'elle soit vraie deux à deux pour tous les processus ayant eu une communication directe. Il existe cependant une autre méthode pour caractériser la cohérence d'un état global, qui considère aussi les relations indirectes entre processus. Elle a été proposée par Netzer et Xu dans et est basée sur la notion de chemin **« zig-zag »**, une généralisation de la relation de causalité de Lamport. Comme, on peut le voir dans la figure 6.12, C_P^1 n'est pas en relation causale avec C_R^1 , pourtant ils ne peuvent pas appartenir tous deux à un même état global cohérent, puisque soit le message $m1$ dans l'état global$[C_P^1, C_Q^2, C_R^1]$, soit le message $m2$ dans l'état global $[C_P^1, C_Q^1, C_R^1]$ serait orphelin.

Cette relation qui existe entre C_P et C_R se nomme chemin « zig-zag ». C'est une extension de la relation de causalité de Lamport et peut se définir comme:

Définition 6.4.7 Il existe un chemin« zig-zag » entre deux points de reprise C_P et C_Q si et seulement si il existe des messages m_1, m_2, ..., m_n tels que :

m_1 est émis par P après la prise de C_P

si $m_k (1 < k < n)$ est reçu par le processus R, alors le message $m_k + 1$ est envoyé par R dans le même intervalle de points de reprise (ou dans un intervalle suivant). $m_k + 1$ peut être envoyé aussi bien avant qu'après la réception de m_k.

m_n est reçu par Q avant la prise de C_Q

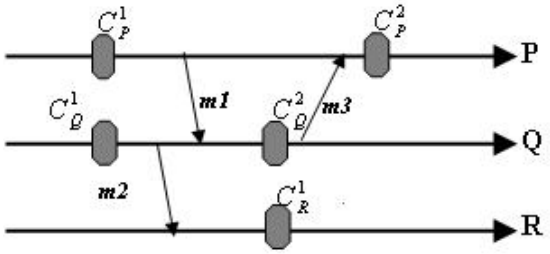


FIGURE 6.12 – Chemin « zig-zag » entre C_P et C_R

Lorsqu'il existe un chemin « zig-zag » entre un point de reprise et luimême, on parle alors d'un cycle « zig-zag ». De fait, un point de reprise pris dans un cycle « zig-zag » ne peut biensûr appartenir à aucun état global cohérent, et est donc inutile. Par exemple, dans la figure 6.12 le point C_Q^2 est pris dans un cycle « zig-zag » à cause des messages $m1$ et $m3$.

Définition 6.4.8 **La cohérence forte** étend la cohérence simple, avec la condition supplémentaire qu'il n'existe pas de messages en transit dans l'état global considéré.

6.4.3 Les différents protocoles

Nous présentons ici les différentes familles de protocoles de tolérance aux pannes par points de reprise : indépendants, synchronisés et induits par messages.

Indépendant

Chaque processus sauvegarde de manière indépendante son état local dans un point de reprise, il en conserve plusieurs. Durant l'exécution, les messages sont accompagnés d'une estampille temporelle qui permet de percevoir des dépendances causales entre les états des différents processus (voir la figure 6.14). Lors de la reprise, la ligne de recouvrement est calculée à l'aide des estampilles temporelles. Le processus défaillant recouvre à partir de l'un de ses points de reprises. Les autres processus peuvent ou non être amenés à faire un recouvrement arrière, en fonction des dépendances causales.

Cette technique a comme principal avantage que chaque processus sauvegarde son point de reprise quand cela est le plus avantageux pour lui-même. Ainsi, il est possible pour un processus de faire cette opération quand la taille des informations sur son état est petite, minimisant ainsi la taille du point de reprise et limitant la dégradation des performances lors d'une exécution sans fautes.

Cependant, cette technique possède différents inconvénients. Premièrement, le calcul de la ligne de recouvrement peut s'avérer coûteux, surtout si le nombre de processus est grand et les communications importantes. De plus, il y a un risque d'effet domino qui a pour conséquence de ramener l'application dans son état initial, perdant ainsi tout le calcul déjà effectué.

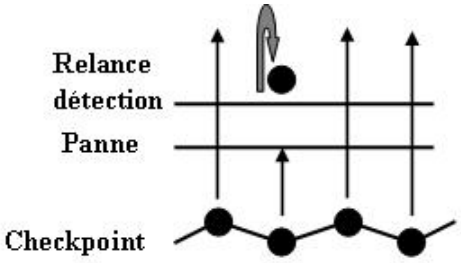


FIGURE 6.13 – Le mécanisme de checkpoint Indépendant

Synchronisé

Cette approche repose sur une coordination globale de tous les processus de l'application. Un processus, le coordinateur (désigné dynamiquement ou statiquement) sauvegarde son point de reprise et diffuse un message **CKPT(checkpoint)** demandant à tous les autres processus de l'application d'établir leur point de reprise. Quand un processus reçoit le message, il stoppe son exécution, vide ses canaux de communication, prends un point de reprise provisoire, et envoie un message d'acquittement au coordinateur. Une fois que le coordinateur a reçu l'ensemble des acquittements, celui-ci diffuse un message de validation du point de reprise. Chaque processus remplace alors, de manière atomique, son ancien point de reprise permanent par le point de reprise provisoire.

Lors de la reprise, il suffit de restaurer l'ensemble des processus de l'application à partir de leur point de reprise respectif. Cette technique assure que l'ensemble des points de reprise forme un état global cohérent et évite ainsi l'effet domino. De plus, la reprise est très simple. Un autre avantage de cette technique est que l'espace de stockage nécessaire pour conserver les points de reprise est minimisé puisqu'il n'est nécessaire que d'en conserver un seul par processus.

Le principal inconvénient de cette stratégie est la latence importante qu'elle implique lors de la sauvegarde du point de reprise, étant donné qu'elle nécessite une coordination globale de tous les processus.

Deux problèmes se posent donc ici, la perte de performance y compris en l'absence de fautes, et la gestion du passage à l'échelle. En effet, plus le nombre de processus augmente, plus la latence risque d'être importante, chaque processus devant attendre la fin de l'opération avant de pouvoir reprendre son exécution normale.

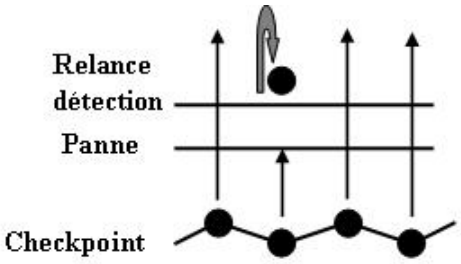


FIGURE 6.14 – Le mécanisme de checkpoint Synchronisé

Induits par messages

La synchronisation se fait ici de manière "paresseuse", en utilisant les messages de l'application. Chaque processus prend régulièrement des points de reprise de manière indépendante.

Cependant, en fonction des messages reçus et envoyés et des informations qui sont transportées sur ces messages, le processus devra peutêtre prendre un point de reprise additionnel ; on dit que cette prise est forcée.

L'algorithme de décision assure (ou maximise la probabilité) que le point pris fasse partie d'un état global cohérent, et donc qu'il existe une ligne de recouvrement toujours assez récente. On distingue deux familles de protocole:

— les protocoles (model-based) : les communications et les prises de points de reprise doivent respecter un certain motif : par exemple, si tout envoi et toute réception de message est précédé d'un point de reprise, alors tous les points au moins à un état global cohérent, et le dernier point pris fait toujours partie de la ligne de recouvrement.
Ces protocoles sont généralement basés sur les notions de chemins et de cycles "zig-zag" pour déterminer les états globaux cohérents.

— les protocoles (index-based) : les points de reprise sont indexés, chaque message porte l'index du dernier point de l'émetteur. Sur réception d'un message indiquant un index supérieur au sien, le récepteur doit prendre un point de reprise avant la prise en compte du message : l'incohérence est évitée "au dernier moment". Ce type de protocole utilise donc une méthode basée sur la suppression des dépendances causales entre les points de reprise des processus qui ont des communications directes.

6.5 Résumé du chapitre

La tolérance aux pannes est un sujet important dans la conception de systèmes distribués. La tolérance aux pannes est définie comme la caractéristique par laquelle un système peut masquer la l'occurrence et le recouvrement après les pannes. En d'autres termes, un système est tolérant aux pannes s'il peut continuer à fonctionner en présence de pannes.

La redondance est la technique clé nécessaire pour atteindre la tolérance aux pannes. Lorsque appliquée aux processus, la notion de groupes de processus devient importante. Un groupe de processus se compose d'un certain nombre de processus qui coopèrent étroitement pour fournir un service.

Dans les groupes de processus tolérants aux pannes, un ou plusieurs processus peuvent échouer sans affecter la disponibilité du service mis en œuvre par le groupe. Souvent, il faut que la communication au sein du groupe soit hautement fiable et adhère à des propriétés d'ordre et d'atomicité strictes afin d'obtenir une tolérance aux pannes.

La recouvrement dans les systèmes tolérants aux pannes est invariablement réalisée en contrôlant régulièrement l'état du système. Ce contrôle est complètement distribué et c'est une opération coûteuse. Pour améliorer les performances, de nombreux systèmes distribués combinent le point de reprise avec la journalisation des messages.

En enregistrant la communication entre les processus, il devient possible de restaurer l'exécution du système après un crash.

6.6 Exercices

Exercice 6.1 Considérez l'algorithme de point de reprise simple suivant. Un processus prend un point de reprise local juste après l'envoi d'un message. Montrez que le dernier point de reprise à tous les processus sera toujours cohérent. Quels sont les compromis avec cette méthode ? ■

Exercice 6.2 Trois ordinateurs fournissent ensemble un service répliqué. Les fabricants prétendent que chaque ordinateur a un délai moyen entre pannes de cinq jours ; une panne prend généralement quatre heures pour la réparer. Quelle est la disponibilité du service répliqué ? ■

Bibliographie

Livres

[Lam78] Leslie LAMPORT. *Time, Clocks, and the Ordering of Events in a Distributed System*. T. 21. dans Communications of the ACM, 1978, p. 558-565.

[Tan95] Andrew TANENBAUM. *Distributed operating systems*. 5e éd. Prentice Hall International, 1995.

[Ben05] Taha BENNANI. *Tolérance aux fautes dans les systèmes répartis à base d'intergiciels réflexifs standards*. INSA de Toulouse, 2005.

[Abi07] Sedjal Halima ABID SABRINA. *Développement d'une distribution verticale tolérante aux pannes pour le dataming*. Université da Mascara– Département Informatique, 2007.

[Bou07] Khaldi Ibrahim BOUFERA HADIYA. *Intégration des mécanismes de tolérances aux pannes dans les applications Data Mining Distribué*. Université da Mascara– Département Informatique, 2007.

[Ta07] Andrew TANENBAUM et AL. *DISTRIBUTED SYSTEMS (Principles and paradigms)*. 2e éd. Vrije university Amdterdam, The Netherlands : Pearson Education. Inc., 2007.

[Aja08] Mukesh Singhal AJAY D. KSHEMKALYANI. *DISTRIBUTED COMPUTING (Principles, algorithms, and systems)*. 2e éd. Vrije university Amdterdam, The Netherlands : Cambridge university press, 2008.

[Car08] Eric CARIOU. *Systèmes distribué, Introduction générale*. 2e éd. Université de Pau et des Pays de l'Adour UFR Sciences Pau – Département Informatique, 2008.

[Ca12] George COULOURIS et AL. *DISTRIBUTED SYSTEMS (concepts and design)*. 5e éd. Addison Wesley, 2012.

[Sai12] Limam SAID. *Gestion de tolérance aux fautes dans les cloud computing avec clous sim*. Université d'oran– Département Informatique, 2012.

Articles

[Fly72] M. FLYNN. « Some Computer Organizations and Their Effectiveness ». In : C 21 (1972), p. 948.

En ligne

[Ben] Badr BENMAMMAR. *Algorithmique des Systèmes et applications Réparties (horloges logiques*. URL : https://slideplayer.fr/slide/13095754/. (accessed: 29.01.2021).

[Coh] Johanne COHEN. *Horloges logiques et datation des évènements. LORIA/CNRS, Nancy, France.* URL : https://www.lri.fr/~jcohen/documents/enseignement/CoursHorloge.pdf. (accessed: 29.01.2021).

[Hut] Guillaume HUTZLER. *Systèmes Distribués (1)*. URL : https://www.ibisc.univ-evry.fr/~hutzler/Cours/SyDi/SyDiIntroduction.pdf. (accessed: 10.12.2020).

[Kai] C. KAISER. *Systèmes et Applications Répartis Ordres, état global, horloges, synchronisation, contrôle et reprise dans les systèmes répartis*. URL : http://deptinfo.cnam.fr/Enseignement/CycleSpecialisation/SAR/Poly_C_Kaiser.pdf. (accessed: 27.01.2021).

[Kra] Sacha KRAKOWIAK. *Consensus - 1 Systèmes synchrones et asynchrones avec pannes franches*. URL : https://docplayer.fr/37435897-Consensus-1-systemes-synchrones-et-asynchrones-avec-pannes-franches.html. (accessed: 02.02.2021).

[Ma] MEHREZBOULARES et AL. *SystèmesRépartis*. URL : https://docplayer.fr/1912974-Systemes-repartis-mr-mehrez-boulares-mr-nour-ben-yahia-2013-2014.html. (accessed: 29.01.2021).

[MUD] Juslin TSHIAMUA MUDIKOLELE. *Mise en oeuvre d'un système distribué pour l'identification et le suivi du casier judiciaire*. URL : https://www.memoireonline.com/02/17/9572/m_Mise-en-oeuvre-dun-systeme-distribue-pour-lidentification-et-le-suivi-du-casier-judiciaire.html#google_vignette. (accessed: 09.12.2020).

[PHI] Laurent PHILIPPE. *Algorithmique Distribuée Introduction et Contexte*. URL : https://members.femto-st.fr/sites/femto-st.fr.Laurent-Philippe/files/content/lessons/introduction.pdf. (accessed: 23.12.2020).

Index