République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université Ahmed Draia - Adrar

Faculté des Sciences et de la Technologie

Département des Mathématiques et Informatique

Mémoire de fin d'étude, en vue de l'obtention du diplôme de Master en informatique

Option: Réseaux et Systèmes Intelligents

Thème

Un algorithme distribué auto stabilisation pour la résolution d'un système d'équation linéaire

Préparé par

OMARI Youcef
BEN BRAHIM Youcef

Soutenu le 22 Mai 2016, devant le jury composé de:

Dr. OMARI Mohammed	Président
Mr. KOUHILI Mohammed	Encadreur
Mr. KADI Mohammed	Examinateur
Mr. MAMOUNI Elmamoun	Examinateur

Année Universitaire 2015/2016.

DEDICACES

A celui qui m'a indiqué la bonne voie en me rappelant que la volonté fait toujours les grands Hommes

A mon Père,

A celle qui a attendu avec patience les fruits de sa bonne éducation,...

A ma Mère,

A ceux qui ont fait preuve de soutiens, et qui m'ont donné une motivation sans prix

A mes frères,

A tous ceux qui ont cru en mes succès...

Ben Brahim Youcef

Ce travail est dédié à:

Mes parents;

Mes frères et sœurs;

Et tous mes amis.

Omari Youcef

Remerciement

Nous remercions le seigneur tout puissant de nous avoir accordé volonté et patience dans l'accomplissement de ce travail à terme.

Nous tenons à exprimer nos vifs remerciements à ***Mr. Kouhili Mohammed,*** *Professeur de l'informatique à l'université d'Adrar pour avoir accepté de nous encadrer et de nous diriger, aussi pour sa patienté, pour ses conseils précieux et pour toutes les commodités et aisances qu'il nous a apportées durant notre étude et réalisation de ce projet.*

Sans oublier bien sûr de remercier profondément tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail.

OMARI YOUCEF et BEN BRAHIM YOUCEF

Résumé

Nous avons présenté dans ce mémoire un état de l'art des systèmes distribués, qui a pour objectif d'utiliser et implémenté un algorithme distribué pour la résolution d'un système d'équation linéaire par la méthode gauss. Les systèmes distribués ont résolus beaucoup de problèmes, et avec l'évolution de la technologie du monde actuel, le système distribué joue un rôle important dans les différents domaines (scientifiques, commerciales,...). Il est devenu classique de s'appuyer sur multiples unités distribuées pour améliorer la performance d'une application, la tolérance aux pannes, ou pour traiter des problèmes dépassant les capacités d'une seule unité de traitement. Les propriétés attendus de tels systèmes sont la sûreté, la fiabilité, la disponibilité, l'intégrité, la confidentialité, etc.

Dans une approche plus pratique, nous exposons nos contributions dans le développement de la plateforme logicielle ViSiDiA pour la simulation et la visualisation d'algorithmes distribués. La simulation offre la possibilité d'étudier les performances des applications distribuées sans la complexité et le coût des plates-formes d'exécution réelles.

Nos principales contributions sont la mise en place un algorithme distribué de manière telle que dans le cas de panne, le système continue à fonctionner de façon acceptable pendant que les réparations sont en cours. En d'autres termes, le système distribué doit être tolérant aux pannes. Cela signifie que les défaillances partielles n'affectent pas sérieusement la performance globale du système.

Pour atteindre cela, notre algorithme se compose de deux étapes principales; la première étape on a d'abord mise en place un algorithme qui permet de construire l'arbre sans cycle et couvrant de poids minimal dans notre système; On a inspiré le principe de notre algorithme de la méthode d'arbre couvrant de poids minimal. Ensuite pour la deuxième étape, on a implémenté un mécanisme qui permettre d'assurer la tolérance aux pannes, cela veut dire un mécanisme qui permettre à le système de fonctionner même en présence de la panne de l'un de leurs composants.

Mots clés:

Système distribué, algorithmes distribués, Système d'équations linéaires, tolérance aux pannes, ViSiDiA.

Sommaire

CHAPITRE II Le problème des pannes en systèmes distribués

CHAPITRE III Implémentation et Résultats expérimentaux

Liste des figures

Liste des tableaux

Introduction générale

Les systèmes informatiques (Ordinateurs) ont connu une révolution. Dès 1945, lorsque l'ordinateur moderne a commencé, jusqu'à 1985 environ, les ordinateurs étaient volumineux et coûteux. Même les mini-ordinateurs coûtent au moins des dizaines de milliers de dollars chacun. En conséquence, la plupart des organisations avaient seulement une poignée d'ordinateurs, et par l'absence d'un moyen de les relier, ceux-ci fonctionnent indépendamment les uns des autres.

À partir d'environ le milieu des années 1980, cependant, deux avancées dans la technologie ont commencé à changer cette situation. Le premier était le développement de microprocesseurs puissants. Le second développement a été l'invention des réseaux informatiques à grande vitesse.

Depuis l'apparition des réseaux informatiques, nos besoins en termes de calcul et de communication augmentent de jour en jour. Alors un système appelé centralisé, basé sur une seule machine fait son apparition et qui a résolu beaucoup de problèmes, Mais l'augmentation journalière des besoins a contribué à l'émergence de l'informatique dite répartie, pour répondre à ce problème.

L'informatique répartie est l'un des enjeux majeurs de l'informatique à l'heure actuelle et dans le futur. Nous sommes en train de passer d'une architecture où une machine fournissait des services à un ensemble de machines (système centralisé), à une architecture où un ensemble de machines reliées par un réseau, compose un système qui fournit des services (système réparti).

Ces systèmes sont devenus un élément déterminant de la compétitivité de tous les secteurs économiques et même de l'évolution de la société par leur impact dans des domaines comme le commerce, la sante, l'éducation, les loisirs, l'environnement, etc.

Un système distribué est un ensemble d'entités autonomes (ayant une mémoire et une capacité d'exécuter). Ces entités partagent des informations et collaborent par échange de messages et coopèrent pour résoudre un même problème. Les entités peuvent représenter tout un tas d'objets différents: des téléphones portables, des tablettes, des processus, des ordinateurs, des robots, des drones, ou même un être humain. Pour simplifier, on choisira à partir de maintenant le terme de « processeur » comme synonyme d'entité de calcul, bien que

ce dernier terme soit plus générique. En effet, les processeurs peuvent être géographiquement très éloignés les uns des autres.

La particularité des systèmes distribués, et ce en quoi ils différent de certains autres systèmes qui sont aussi répartis sur plusieurs entités, est qu'il n'y a pas de centralisation. Les processus du système n'ont qu'une connaissance partielle de l'état du système, et ne peuvent effectuer que des actions locales, en ayant pourtant un objectif global. Ainsi, l'accent est mis sur la méthode collaborative pour résoudre une tâche donnée.

Dans un système distribué, tous les processeurs exécutent le même algorithme et celui-ci doit fonctionner quel que soit la topologie du réseau. Un algorithme distribué est une collection d'algorithmes locaux, chacun de ces algorithmes étant exécuté par un processus du système et la communication entre les processus se fait par échange de messages uniquement. Un algorithme distribué est donc conçu pour s'exécuter sur un système distribué et décrit une collection de comportements distincts. Il doit être correct: résoudre le problème pour lequel il est conçu, efficace: minimiser les ressources utilisées et tolérant aux pannes: récupérer un comportement normal après une panne. Dans un tel contexte, la sûreté de fonctionnement des applications est un élément de première importance et un mécanisme de tolérance aux pannes devient nécessaire pour en assurer certains aspects. L'une des caractéristiques des systèmes distribués qui les distinguent des systèmes centralisés est la notion de panne partielle: une partie du système est défaillant alors que la partie restante continue à fonctionner, et apparemment correctement.

L'objectif de notre travail est d'implémenter et de réaliser un algorithme distribué pour la résolution d'un système d'équation linéaire par la méthode gauss. Mais pour être plus précis, nous visons à mettre en place un algorithme distribués de manière telle qu'il peut récupérer automatiquement des défaillances partielles sans affecter sérieusement la performance globale. En particulier, en cas de panne le système devrait continuer à fonctionner de façon acceptable pendant que les réparations sont en cours. En d'autres termes, le système distribué doit être tolérant aux pannes. Ces systèmes sont très complexes à développer et à vérifier; la modélisation et la spécification de tels systèmes distribués est un enjeu pour assurer la qualité de tels systèmes.

La structure du notre mémoire s'articule autour de trois chapitres:

Nous présentons dans le premier chapitre de ce mémoire, notions de base des systèmes et algorithmes distribués, les différents composants qui les constituants et nous expliquons ensuite leurs caractéristiques. Nous présentons aussi, les définitions et les principaux concepts liés aux systèmes distribués, les différents modes de communication, la Complexité et les problèmes des systèmes et des algorithmes distribués.

Au deuxième chapitre, nous présentons les différents types des pannes qui peuvent affecter le système distribué. Ainsi que, on définit la tolérance aux pannes et les techniques qui permettre d'assurer la tolérance aux pannes.

Nous présentons dans le troisième chapitre une description détaillée de notre algorithme distribué pour la résolution d'un système d'équation linéaire avec l'explication des idées et des mécanismes qui on a utilisé pour détecter et résoudre les problèmes des pannes dans ce algorithme distribué. Ensuite, Nous définissons les différents ressources et techniques qui on a utilisé dans notre projet. Ainsi que nous présentons les différents résultats d'expérimentations obtenus par notre algorithme en utilisant le simulateur VISIDIA. Enfin Le mémoire se termine, par une conclusion générale.

Chapitre I

Les systèmes et les algorithmes distribués

1. Introduction

Un système distribué est constitué d'une collection d'entités du calcul autonomes (ordinateurs, périphériques...) qui échangent des messages et reliés entre eux via un système de communication.

Les systèmes distribués ont connu une croissance phénoménale au cours des dernières années. La baisse du coût du matériel, les progrès de la technologie de communication, l'évolution de la culture des entreprises dans un marché mondial concurrentiel sous divers aspects (Commerce, Marketing, Bancaires, ...), le besoin d'intégration des applications existantes initialement séparées ainsi que l'intégration massive de ressources, l'évolution des besoins des utilisateurs (Qualité de service, Communication et partage des informations, Ouverture vers d'autres systèmes) **[02],** et finalement notre dépendance sans cesse croissante sur les réseaux pour une vaste gamme d'applications allant de la communication sociale à les transactions financières ont contribué à cette croissance.

Les systèmes distribués sont plus complexes à appréhender que les systèmes centralisés et séquentiels car l'ordre des actions est plus difficilement déterminé. Les systèmes distribués se distinguent des systèmes parallèles et concurrents par la distribution intrinsèque des données et des unités de transformation de ces données. La répartition des données et des unités de calcul est subie, et le but est de masquer cette distribution, en offrant aux utilisateurs les outils nécessaires pour l'oublier, et ainsi faciliter l'utilisation du système. [**28]**

La puissance d'un système distribué provient de la possibilité de diviser le travail entre les différentes entités. **[22]** Les systèmes distribués varient des systèmes centralisés dans un certain nombre d'aspects essentiels**:**

- ✓ Le manque de connaissances de l'état global du système: En général, un processus n'a pas de connaissance sur les états locaux des autres processus.
- ✓ L'absence d'une horloge globale: Il n'y a pas d'ordre total sur les événements par rapport à leur apparition dans le temps. Exemple: l'exclusion mutuelle devient un problème.
- ✓ Non-déterminisme: L'exécution des processus est non-déterministe. Exemple: L'exécution d'un système deux fois peut donner des résultats différents. **[26]**

Un algorithme distribué est une collection d'algorithmes locaux, un pour chaque processus dans le système. Ce sont des algorithmes qui impliquent généralement plus d'un

hôte lors de l'exécution; Ceux-ci sont généralement conçus pour fonctionner dans des environnements en réseau. Il est défini par le comportement de toutes les entités distinctes. **[24]**

Dans la suite de ce chapitre, nous présentons la notion générale de systèmes et algorithmes distribués, les différents Composants qui les constituants et nous expliquons ensuite leurs caractéristiques.

2. Définition de système et Algorithme distribué

2.1. Définition de système distribué

Comme toujours, dans le domaine d'informatique, chaque terme possède de nombreuses définitions. Les deux termes « réparti » et « distribué » sont couramment utilisés pour désigner les mêmes concepts fondamentaux. **[04]**

Diverses définitions des systèmes distribués ont été donnés dans la littérature, aucun d'eux satisfaisant, et aucun d'entre eux en accord avec tous les autres.

Coulouris, Dollimore et Kindberg donne la définition suivante: Un système distribué ou réparti est un système dont les composants sont répartis sur différents nœuds d'un réseau d'ordinateurs. Ces composants communiquent et coordonnent leurs actions uniquement par l'échange de messages. **[27]** Le World Wide Web, qui correspond à cette définition, n'est pourtant pas considéré comme un système distribué par **Tanenbaum et Van Steen.** À leur avis, un système distribué doit apparaître auprès de ses utilisateurs comme un seul système cohérent. **[16]** Or les utilisateurs du Web voient bien que les documents qu'ils consultent sont situés à différents endroits et sont gérés par différents serveurs. Ce n'est donc pas à leurs vues un système distribué.

Ces deux exemples montrent qu'aucune définition ne fait l'unanimité. **[01]**

2.1.1. Définition générale: Un système distribué est un ensemble d'ordinateurs indépendants connectés en réseau et communiquant via ce réseau. Cet ensemble apparaît du point de vue de l'utilisateur comme un cohérent unique système, [**34]** Chacun de ces ordinateurs n'ayant qu'une vue partielle et retardée de l'état global courant.

Il y a deux aspects de cette définition. La première concerne le matériel: les machines sont autonomes. La seconde traite avec le logiciel: les utilisateurs pensent qu'ils ont affaire à un seul système. **[24]**

2.2. Définition d'algorithme distribué

Un algorithme est un ensemble d'instructions qui permet de régir le déroulement d'un programme informatique.

On dit algorithme distribué s'il est exécuté de manière simultanée sur un ensemble de ressources. Cette exécution, en simultanée sur plusieurs ressources distinctes, permet alors la réalisation d'un seul et même calcul. Le comportement de chaque processus est déterminé par un algorithme local et la communication entre les processus se fait par échange de messages uniquement. Dans la réalisation d'un calcul, il est possible de rencontrer différents types d'exécution:

1- Linéaire: le calcul est réalisé sur une seule ressource;
2- Distribuée: le calcul est partagé entre toutes les ressources et s'exécute de façon simultanée. **[23]**

3. Principes l'algorithmiques distribuées

Si l'on dispose d'un grand nombre d'ordinateurs, répartis géographiquement sur un vaste territoire et reliés entre eux par un réseau, peut-on utiliser simultanément la puissance de toutes ces machines pour effectuer un même calcul global ?

Réunir un groupe d'ordinateurs reliés en réseau afin de former un système de calcul équivalent à une machine vectorielle de très grande puissance est l'objectif de l'algorithmique distribuée.

Pour l'utilisateur d'un tel système, celui-ci doit être vu comme une seule machine, mono-utilisateur et monoprocesseur, de grande puissance.

Une première ambition de l'algorithmique distribuée consiste à pouvoir considérer qu'aucun ordinateur distribué n'a un rôle différent des autres, c'est-à-dire qu'il n'y a pas de serveurs. Ainsi, tout ordinateur défaillant peut être remplacé par n'importe quel autre ordinateur opérationnel. Pour effectuer de tels remplacements, il convient de disposer

d'algorithmes qui résistent aux défaillances et qui sont capables de maintenir la spécification générale du système même si plusieurs ordinateurs ont un comportement incohérent. **[28]**

4. Un cadre de base pour les systèmes distribués

Un système distribué est un ensemble composé d'entités de calculs reliés par un système de communication; ces entités de calculs peuvent représenter tout un tas d'objets différents: des téléphones portables, des tablettes, des ordinateurs, des satellites, des robots, etc. Pour simplifier, on choisira à partir de maintenant le terme de « processeur » comme synonyme d'entité de calcul, bien que ce dernier terme soit plus générique. **[25]** Aussi, Il existe principalement deux modèles de systèmes distribuées selon la manière de communiquer des processeurs: le « passage de messages » et la « mémoire partagée »:

4.1. Passage de messages

(En Anglais: message passing) Ce modèle traite explicitement des communications. Un processeur communique avec les processeurs voisins par l'envoie et la réception de messages à travers un médium (canaux) de communications adjacents. Le modèle de communication par passage de messages est le modèle couramment utilisé dans les systèmes distribués. Celui-ci est beaucoup plus proche de la réalité.

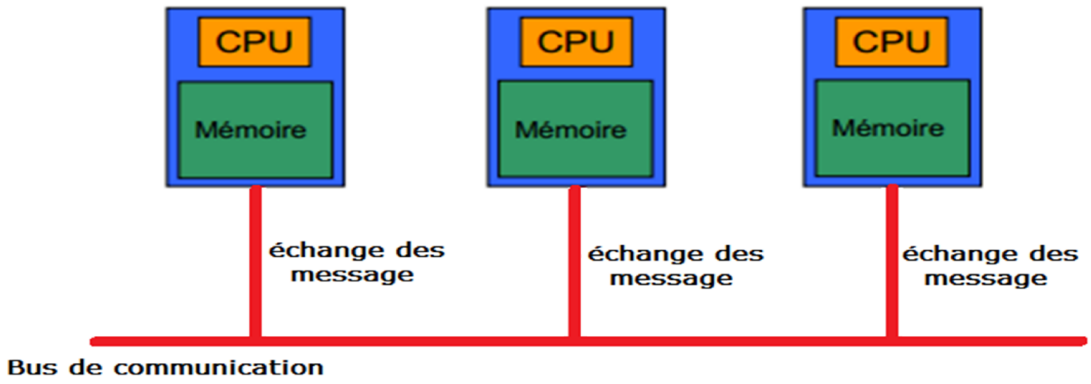


Figure 1.1: Modèle Passage de messages.

4.2. Mémoire partagée

Au contraire, dans ce modèle, les processeurs ne communiquent pas entre eux directement. Ils s'échangent des informations grâce à une mémoire ou des variables

communes qu'ils peuvent tous lire et modifier à l'aide des instructions du type (Lire/Ecrire) **[13]**.

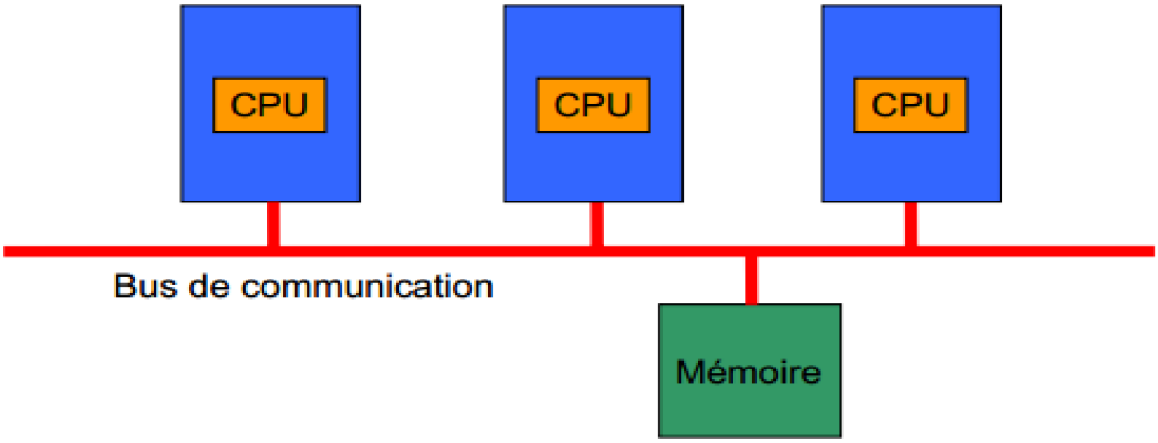


Figure 1.2: Modèle mémoire partagée.

Dans notre travail, on a utilisé le modèle par passage de messages pour implémenter notre système distribué. Il existe plusieurs modes de communications dans ce modèle.

4.3. Réseau de diffusion (broadcast/radio networks)

Chaque processeur peut communiquer le même message simultanément à un certain nombre de récepteurs, comme un émetteur radio le ferait. Là encore il y a de nombreuses variantes suivant que les réceptions multiples en un processeur créent ou non des collisions et si les collisions sont détectables en tant que telles. **[13]**

4.4. Point-à-point

Dans ce mode de communication, chaque processeur ne peut communiquer directement qu'avec un certain nombre de processeurs, ses voisins. On modélise de tels systèmes naturellement par un graphe connexe (voir la figure 1.3 ci-dessous). Les sommets sont les processeurs, et les arêtes sont les canaux de communication. On supposera que le graphe est non-orienté et sans multi-arêtes. **[13]**

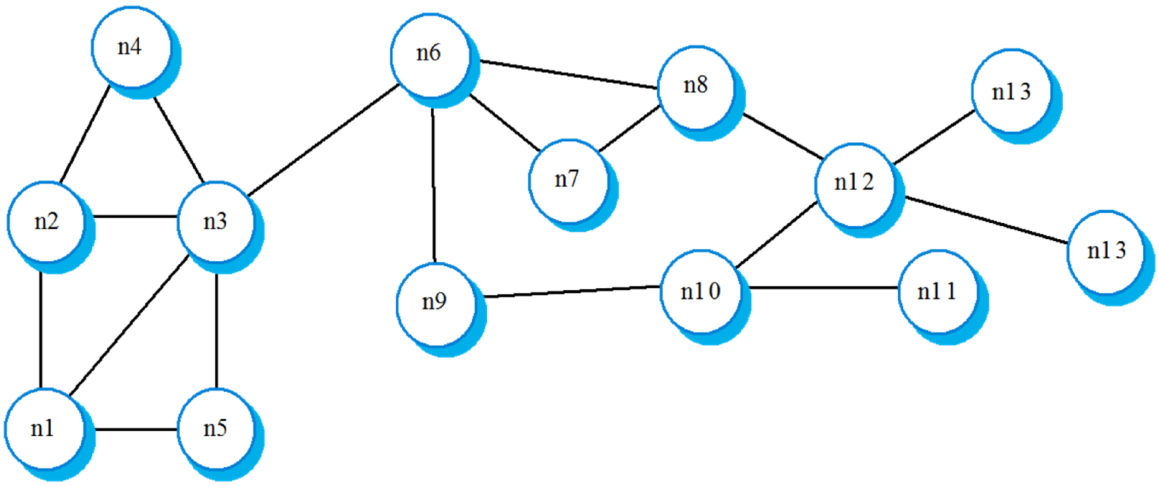


Figure 1.3: Modèle Point-à-point.

5. Systèmes distribués vs parallèles

5.1. Systèmes Parallèles

Un système parallèle est un ensemble de processeurs qui sont proches les uns des autres ou bien une machine multiprocesseur avec un environnement du type SIMD (Single Instruction Multiple Data). Tous les processeurs exécutent le même programme et ont une vision uniforme de l'état global du système. **[12]** De plus, tous les processeurs ont accès à une mémoire commune et partagée par chacun d'eux. Cette mémoire commune permet d'échanger de l'information entre les processeurs. Dans un système parallèle, il est possible de choisir l'algorithme (ou une partie de celui-ci) qui sera exécuté par chaque processeur. L'architecture connectant les processeurs entre eux est prédéfinie. La tâche à résoudre peut être ainsi découpée et répartie sur les processeurs. Le but est d'exploiter au maximum la puissance de calcul du système. **[28]**

Dans les ordinateurs classiques, les architectures 32 bits ou plus par exemple, sont déjà des sortes de machines parallèles dans la mesure où elles traitent 32 bits simultanément (à l'intérieur d'un même cycle machine). Les machines parallèles possèdent un fort degré de synchronisation. **[13]**

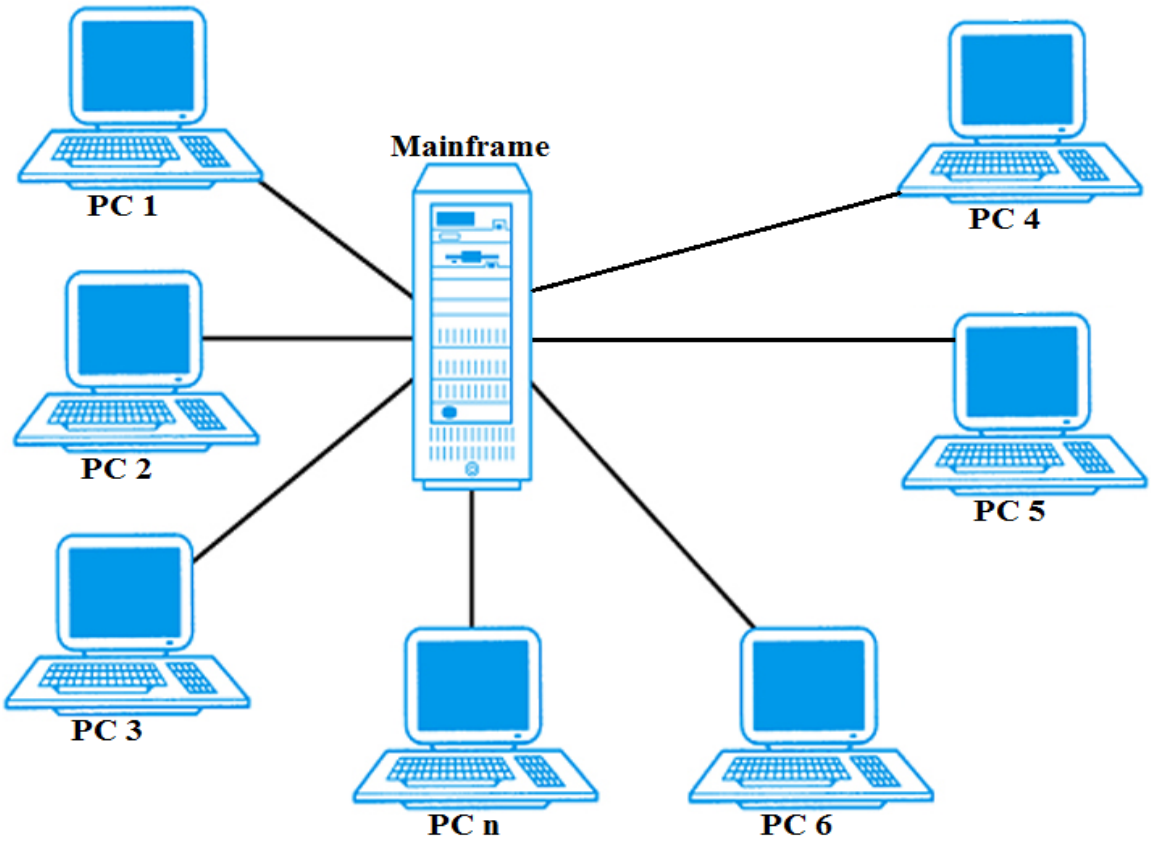


Figure 1.4: Système Parallèle.

5.2. Systèmes distribués

Un système distribué est un ensemble des processeurs indépendants sur des machines distinctes interconnectés par un réseau informatique et communiquant par échange de messages. **[12]** De plus, chaque processeur possède sa propre mémoire locale et échange de l'information avec les autres processeurs à l'aide de messages envoyés via le réseau les connectant. Dans un système distribué, tous les processeurs exécutent le même algorithme et celui-ci doit fonctionner quel que soit la topologie du réseau. Chaque processeur n'a qu'une vision partielle du système et doit communiquer avec les autres processeurs afin d'accomplir sa tâche. Ainsi, l'accent est mis sur la méthode collaborative pour résoudre une tache donnée **[28].**

En résumé, ce qui caractérise les systèmes distribués par rapport aux machines parallèles, c'est **[13]**:

1. Potentiellement beaucoup plus d'entités de calcul (Processeurs);

2. Distance physique entre entités plus importante.

Le calcul distribué est donc l'art de faire des calculs avec de tels systèmes. On fait des calculs avec des systèmes distribués pour essentiellement deux raisons :

1. Augmenter à moindre coût la puissance de calcul et de stockage, en ajoutant graduellement des entités ou en regroupant des ensembles d'entités déjà existantes (clusters ou grappes de PC, Grid5000, ...);
2. Communiquer entre entités distantes via des protocoles (réseaux LAN, réseaux de satellites, Wifi, téléphonie mobile, ...), mais aussi mettre en rapport des banques de données réparties (Internet, transactions bancaires, échange de fichiers peer-to-peer, ...).

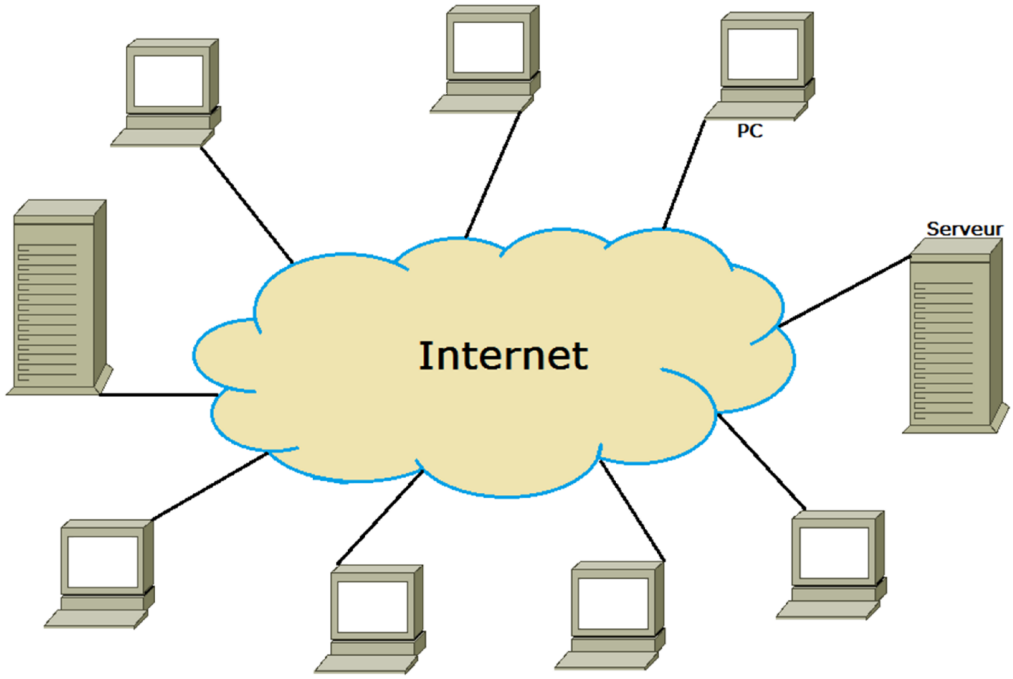


Figure 1.5: Système distribué.

6. Exemples des systèmes distribués

Il existe de nombreux exemples de systèmes distribués qui sont utilisés dans la vie quotidienne dans une variété d'applications. Certains systèmes offrent principalement une variété de services utiles aux utilisateurs. En effet, une application distribuée est un système composé de processus qui s'exécute sur plusieurs ordinateurs connectés par un réseau. Cet ensemble d'ordinateurs connectés est lui-même un système distribué. **[27]**

La plupart des systèmes sont structurés comme des systèmes client-serveur, où la machine serveur est le gardien des données ou des ressources, et fournissent des services à un certain nombre de clients distribués géographiquement. Quelques applications, toutefois, ne reposent pas sur un serveur central ce sont des systèmes pair à pair (ou peer-to-peer) dont la popularité est en recrudescence. Nous présentons ici quelques exemples de systèmes distribués:

6.1. Internet

Aujourd'hui, l'Internet est partout et sous-tendent de nombreux services de la vie quotidienne que nous prenons aujourd'hui pour acquises: la communication, le partage d'informations, les jeux en ligne, e-mail, réseaux sociaux, e-commerce, recherche sur le web etc.

Internet constitue actuellement le système distribué le plus large au monde et ceci sur quasiment tous les plans. Des utilisateurs de n'importe quelle position dans le monde peuvent utiliser les services offerts par le www (world Wilde web), le FTP (file transfert protocole), et autres applications. Les services disponibles peuvent être étendus librement et le système peut être agrandi par l'ajout d'ordinateurs à n'importe quel moment. **[27]**

6.2. Systèmes Pair à Pair (souvent abrégé « P2P »)

Le système Pair à Pair (en Anglais: peer to peer), est un ensemble de techniques permettant de mettre en place un système distribué qui utilise des ressources distribuées pour réaliser une tâche particulière de manière décentralisée. Les ressources sont composées d'entités de calcul (ex: ordinateurs), de stockage de données, d'un réseau de communication, etc. La tâche à exécuter peut être du calcul distribué, du partage de données (ou de contenu), de la communication et collaboration, d'une plateforme de services, etc. La décentralisation, quant à elle, peut s'appliquer soit aux algorithmes, soit aux données, soit aux métadonnées, soit à plusieurs d'entre eux. **[03]**

L'une des avantages des systèmes peer-to-peer est d'être un réseau sans aucune hiérarchie entre les pairs; En d'autres termes tous les nœuds (pairs) sont en général symétriques, c'est-à-dire chaque client est aussi un serveur. Les fichiers passent directement d'un ordinateur à l'autre, il n'y a pas donc besoin d'un ou de plusieurs serveurs disposant d'une bande passante couteuse pour centraliser et mettre à disposition des fichiers.

En particulier, les systèmes de partage de fichiers permettent de rendre les objets d'autant plus disponibles qu'ils sont populaires, en les répliquant sur un grand nombre de nœuds. Cela permet alors de diminuer la charge (en nombre de requêtes) imposée aux nœuds partageant les fichiers populaires, ce qui facilite l'augmentation du nombre de clients et donc le passage à l'échelle en taille des données. **[03]**

6.3. Grilles informatiques (en Anglais: Grid computing)

Une grille informatique est une infrastructure virtuelle constituée d'un ensemble de ressources informatiques potentiellement partagées, distribuées, hétérogènes, délocalisées et autonomes. Une grille est en effet une infrastructure, c'est-à-dire des équipements techniques d'ordre matériel et logiciel. Cette infrastructure est qualifiée de virtuelle, car les relations entre les entités qui la composent n'existent pas sur le plan matériel, mais d'un point de vue logique. D'un point de vue architectural, la grille peut être définie comme un système distribué constitué de l'agrégation de ressources réparties sur plusieurs sites et mises à disposition par plusieurs organisations différentes. Un site est un lieu géographique regroupant plusieurs ressources informatiques administrées de manière autonome et uniforme. Il peut être composé des millions d'ordinateurs à travers le monde ou d'une grappe de machines (cluster). Les sites sont reliés par un réseau WAN. **[11]**

Le but est de résoudre des problèmes de calcul difficiles plus rapidement et à moindre coût que par les méthodes conventionnelles.

Il existe plusieurs projets de grilles qui ont été mis en place aussi bien à des échelles nationales qu'internationales : la grille expérimentale française Grid'5000, la grille de calcul scientifique nord-américain TeraGrid, la grille chinoise CNGrid, la grille Asie-Pacifique ApGrid , etc.

Comme un exemple est le SETI@home (acronyme de Search for Extra Terrestrial Intelligence, pouvant se traduire par SETI à la maison) : est un projet de calcul distribué utilisant des ordinateurs reliés à Internet pour la recherche d'une intelligence extraterrestre (SETI). Il est hébergé par le Space Sciences Laboratory de l'université de Californie à Berkeley et est accessible au public depuis le 17 mai 1999.

Le grand volume de données qui est constamment recueilli des centaines de radiotélescopes doivent être analysés pour en tirer une conclusion sur l'existence éventuelle

d'une vie extraterrestre. Cela nécessite une puissance de calcul massive. Plutôt que d'utiliser des superordinateurs, l'équipe de l'Université de Californie à Berkeley SETI a décidé de tirer parti de la puissance de calcul au ralenti des millions de PC et postes de travail appartenant à vous et moi. Actuellement, environ 40 giga-octets de données sont tirés vers le bas tous les jours par le télescope et envoyés à plus de trois millions d'ordinateurs partout dans le monde à analyser. Les résultats sont envoyés à travers l'Internet. Le système exécute 14 milliards d'opérations en virgule flottante par seconde et a recueilli plus de 500.000 ans de temps de PC en un an et demi. Il devrait normalement coûté des millions de dollars pour atteindre ce type de puissance sur un ou même deux supercalculateurs. **[05]**

7. Caractéristiques d'un système distribué

Comme nous l'avons vu système distribué est un ensemble d'ordinateurs autonomes, qui sont reliés entre eux par l'intermédiaire d'un réseau informatique.

Même si les systèmes distribués se trouvent dans de nombreuses applications; Cependant leur concevoir est une tâche difficile, car de nombreux problèmes doivent être pris en compte lors de sa mise en œuvre. Tout système idéal distribué doit avoir toutes les caractéristiques décrites ci-dessous. Mais il peut ne pas être possible d'incorporer tous, donc en fonction des exigences d'application les caractéristiques requises sont considérées. **[09]**

Nous citerons dans cette section que celles que nous trouvons les plus connexes à notre contexte d'études: la transparence, le passage à l'échelle, la disponibilité et l'autonomie.

7.1. La transparence

Dans les systèmes distribués, la transparence signifie que le système distribué se présenter aux utilisateurs comme se il était seulement un seul système informatique plutôt que comme une collection de systèmes autonomes, qui coopèrent.

Le Manuel de référence ANSA et **l'Organisation internationale de modèle de référence de normalisation pour Traitement distribué ouvert (RM-ODP)** identifient huit formes de transparence **[15]:**

7.1.1. Transparence d'accès: permet aux ressources locales et distantes pour être accessibles en utilisant des opérations identiques.

7.1.2. Transparence à la localisation: Il s'agit de permettre une transparence d'accès aux ressources du système sans une connaissance préalable de leur localisation.

7.1.3. Transparence à la concurrence: correspond au fait qu'un ensemble de machines indépendantes tentent d'accéder simultanément à une même ressource ou à un ensemble de ressources partagées. Il est alors nécessaire de conserver cette ou ces ressources dans un état cohérent. **[01]**.

7.1.4. Transparence (mobilité) de migration: permet le mouvement des ressources et des clients dans un système sans affecter le fonctionnement des utilisateurs et aussi sans influencer le déroulement des applications.

7.1.5. Transparence de la réplication: En vue d'augmenter la fiabilité et améliorer les performances, on a souvent recours à la duplication de certaines ressources (ex. fichiers de base de données). La gestion de cette duplication ne doit pas être perceptible par l'utilisateur. **[27]**

7.1.6. Transparence de défaillance: Elle permet la dissimulation de défauts, permettant aux utilisateurs et des programmes d'application pour compléter leurs tâches malgré les pannes qui peuvent affecter les composants d'un système (composants matériels ou logiciels).

7.1.7. Transparence de la performance: Dans le but d'améliorer les performances en fonction de la charge (i.e. les demandes de services des clients), il devrait être possible de reconfigurer un système sans que cela ne soit perceptible par l'utilisateur. Par exemple, les sites dits miroirs permettent d'améliorer les performances de l'accès à travers Internet, le déroutement vers ces sites doivent être non perceptible pour l'utilisateur.

7.1.8. Transparence de la mise à l'échelle (Scaling transparency): Elle permet au système d'étendre à l'échelle sans influence notable sur les performances des applications et sans modification de la structure du système.

7.2. Passage à l'échelle

Le concept de passage à l'échelle désigne la capacité d'un système à continuer à délivrer avec un temps de réponse constant un service même si le nombre de clients ou de données augmente de manière importante. Le passage à l'échelle peut être mesuré avec au moins trois dimensions:

1. Le nombre d'utilisateurs et/ou de processus (passage à l'échelle en taille);
2. La distance maximale physique qui sépare les nœuds ou ressources du système (Passage à l'échelle);
3. Le nombre de domaines administratifs (passage à l'échelle administrative) **[01]**.

7.3. Prise en charge des pannes partielles

Les pannes dans un système distribué peuvent être de différente nature et affecter divers composants. Une panne peut être franche, transitoire, intermittente ou byzantine:

- Une panne franche correspondra généralement à un crash, c'est-à-dire que le processus cessera de répondre à toute requête ou d'émettre des requêtes;
- Une panne transitoire ou intermittente, au contraire, la panne apparaîtra à un instant donné puis le processus fonctionnera à nouveau correctement;
- Les pannes byzantines sont caractérisées par un comportement déviant du processus affecté: ce dernier ne respecte plus sa spécification, soit à cause d'une opération malveillante, soit en raison d'une erreur logicielle ou encore à la suite d'une erreur physique non détectée comme une erreur mémoire.

Quel que soit le type de panne qui apparaisse, une première difficulté consiste à la détecter avant de pouvoir la gérer **[01]**.

7.3.1. Détection des pannes

En général, un mécanisme de détection de panne dans un système distribué décrète un délai – time out (retard) – au-delà duquel, si aucune réponse n'a été reçue, le processus est considéré comme étant en panne. Dans un système synchrone pour lequel le délai maximum de communication et de traitement d'une requête est connu, ce système détectera les pannes correctement. Cependant, dans un système totalement asynchrone et pour lequel les délais de réponse ne sont pas prévisibles, comme c'est le cas pour la plupart des systèmes distribués, cette méthode ne permettra pas de distinguer un processus en panne d'un processus très lent.

Ce phénomène abouti à l'impossibilité de la résolution de certains problèmes en présence d'une panne, notamment celui du consensus. Dans lequel un ensemble de processus doivent se mettre d'accord sur une valeur commune **[01]**.

7.4. Disponibilité

Un système est dit disponible s'il est en mesure de délivrer correctement le ou les services de manière conforme à sa spécification. Pour rendre fonctionnement. En effet, l'indisponibilité d'un système peut être causée par plusieurs sources parmi lesquelles nous pouvons citer:

- Les pannes qui sont des conditions ou évènements accidentels empêchant le système, ou un de ses composants, de fonctionner de manière conforme à sa spécification;
- Les surcharges qui sont des sollicitations excessives d'une ressource du système entraînant sa congestion et la dégradation des performances du système **[01]**.

Un système disponible il faut donc le rendre capable de faire face à tout obstacle qui peut compromettre son disponibilité.

7.5. Autonomie

Un système ou un composant est dit autonome si son fonctionnement ou son intégration dans un système existant ne nécessite aucune modification des composants du système hôte. L'autonomie des composants d'un système favorise l'adaptabilité, l'extensibilité et la réutilisation des ressources de ce système. Par exemple, une ressource autonome peut être remplacée avec une autre ressource plus riche en termes de fonctionnalités, ce qui étend les services du système [**01**].

La confiance entre les différentes machines d'un système distribué est un élément essentiel. Nous ajoutons donc cette caractéristique aux cinq précédentes. Dès lors que l'ensemble des machines participant à un système distribué ne sont pas toutes gérées et administrées par la même organisation, des problèmes de confiance entre ces machines apparaissent. Il est nécessaire de proposer des mécanismes pour gérer cette confiance et assurer une certaine protection en son absence [**01**].

7.6. La mémoire répartie

La mémoire répartie est une caractéristique constitutive des systèmes distribués. Chaque machine participant à un système distribué possède sa propre mémoire, son propre espace d'adressage et éventuellement ses propres périphériques de stockage permanents. La mémoire de l'ensemble du système est donc répartie dans le sens où une partie des données se trouve sur un site, une autre partie sur un autre et certaines données peuvent éventuellement être

dupliquées. La prise en charge de cette répartition implique la mise en œuvre de mécanismes de communication pour l'échange de données entre les différents sites. Cet échange peut être explicite, par l'utilisation d'une bibliothèque de communication. Il peut être abstrait dans la notion d'appel de routine à distance. Il peut enfin être implicite dans le cas d'une mémoire virtuellement partagée [**01**].

7.7. Echange de messages

L'échange de messages est vraisemblablement la méthode le plus basse niveau, ou en tout cas la plus proche du modèle d'exécution sous-jacent. Pour qu'une machine **A** connaisse une information contenue sur une machine **B**, cette dernière doit envoyer un message contenant cette information à la machine **A**. Cet envoi de message est explicite au niveau du paradigme de programmation et doit être pris en charge par le développeur [**01**].

8. Les Types des communications

8.1. Communication Synchrones

Dans les types communication Synchrones, Chaque processus $\boldsymbol{p}$ dans le réseau représente une entité qui est capable d'exécuter des pas de calcul, en envoyant et en recevant des messages par ses ports. Nous considérons des systèmes synchrones et un réveil synchrone des processus: les processus procèdent par rondes et commencent l'algorithme en même temps.

Une action de chaque processus se réalise en une suite de pas discrets appelée **ronde** où $\boldsymbol{action_i}$ est un cycle Envoyer / Recevoir / calcul. Plus précisément, Dans une $\boldsymbol{ronde_i}$, des processus envoient des messages (zéro ou plusieurs) à leurs voisins, reçoivent des messages (zéro ou plusieurs) de leurs voisins et effectuent un ensemble de calculs locaux. Les messages envoyés sont reçus pendant la même ronde, c'est-à-dire, si le processus $\boldsymbol{p}$ envoie un message au processus $\boldsymbol{q}$ à la $\boldsymbol{i_{ème}}$ ronde, alors le message est reçu par $\boldsymbol{q}$ pendant la $\boldsymbol{i_{ème}}$ ronde de $\boldsymbol{q}$. **[29]**

Le modèle synchrone, surprenant au premier abord, est une transposition à l'informatique du modèle utilisé par l'électronicien lors de la conception des circuits numériques:

L'établissement des potentiels électriques dans les circuits est considéré comme non observable à L'échelle de l'horloge du système, le basculement de toutes les portes logiques est considéré comme instantané aux instants défini par l'horloge **[30]**.

Ce modèle conduit principalement à deux notions:

1- La simultanéité exprimant le fait que deux événements sont présents en même temps.

2- L'atomicité des réactions exprimant le fait que les actions sont produites dans le même instant que l'apparition des événements qui les ont déclenchées **[30]**.

8.2. Communication Asynchrones

Dans les types communication asynchrones le temps est considéré comme continu, il n'existe donc pas de notion de simultanéité. Il n'existe pas non plus de notions d'atomicité. On s'intéresse ici à la durée des calculs, il est donc possible qu'un événement déclenche une réaction alors que la réaction déclenchée par un événement survenu plutôt n'est pas encore achevée. Ceci conduit à un chevauchement temporel des deux réactions. Ce problème d'exécution simultanée est résolu à l'implantation par des techniques de préemption des calculs. La préemption peut poser des problèmes d'indéterminisme (l'occurrence d'un événement peut conduire à plusieurs exécutions différentes du même programme) et il peut être difficile de borner les temps d'exécution et donc impossible de garantir le respect des contraintes temps réel **[28].**

En pratique il est beaucoup plus difficile de mettre au point un programme écrit pour le mode asynchrone, car on ne sait jamais vraiment dans quel ordre vont s'effectuer la réception des messages (et du coup dans quelle ordre vont s'effectuer les actions correspondantes). Évidemment, en mode synchrone, après la ronde ***i*** vient nécessairement la ronde ***i+1***, et on contrôle de manière beaucoup plus sûre la succession des actions. **[13]**

9. Types des systèmes distribués

Il existe trois grandes familles de systèmes distribués:

1. Les systèmes distribués architecturés autour de systèmes d'exploitation orientés réseaux.

2. Les systèmes distribués reposants sur des systèmes d'exploitation auxquels sont ajoutées des fonctionnalités réparties. Ces systèmes permettent de gérer, de façon plus ou moins évidente, l'hétérogénéité des différents hôtes.

3. Les systèmes distribués reposant sur système d'exploitation complètement distribué. Dans ce cas, le noyau lui-même possède tous les outils pour prendre sous sa

responsabilité la communication inter processus synchrone. Ces systèmes ne gèrent pas directement l'hétérogénéité des différents hôtes **[04].**

10. Mesures de complexité

La complexité d'un algorithme distribué dépend de l'algorithme. La consommation des ressources dans un calcul d'un algorithme distribué peut se mesurer de différentes manières. **[26]**

10.1. Complexité de messages

La complexité en messages d'un algorithme distribué est le nombre total de messages échangés durant l'exécution de l'algorithme jusqu'à la terminaison de celui-ci.

En général pour l'analyse de la complexité en messages d'un algorithme, on fait l'hypothèse qu'un message traverse un lien de communication en une unité de temps. **[28]**

10.2. Complexité en espace

La quantité de mémoire nécessaire par l'ensemble de processus à l'exécution de l'algorithme.

10.3. Complexité en temps

La complexité en temps est le nombre maximal de rondes nécessaires pour que tous les processeurs aient terminé leur calcul. **[29]**

Comme nous l'avons mentionné précédemment, une ronde (un cycle) d'un processeur est composée de trois pas:

1- Envoyer un message à certains voisins;
2- Recevoir un message de certains voisins;
3- Calculer localement.

L'analyse de la complexité en temps des algorithmes proposés utilise l'hypothèse de Synchronisme. C'est une hypothèse forte, néanmoins elle permet d'obtenir une "bonne" Estimation du temps d'exécution de l'algorithme si le système est asynchrone **[28]**.

10.4. Complexité en bits

Par Définition la complexité en bits (pour un canal) d'un algorithme distribué est le nombre de bits qui transitent (par un canal) pendant son exécution. **[28]**

Une ronde d'un bit est une ronde telle que chaque processus peut envoyer/recevoir au plus 1 bit à/de chaque voisin. On définit la complexité en bits d'un algorithme ***A*** comme le nombre de rondes d'un bit nécessaire pour mener à terme l'algorithme. **[29]**

10.5. Complexité en mémoire

La complexité en mémoire est définir comme la taille en nombre de bits nécessaires pour stocker les variables d'un algorithme exécuté sur chaque processeur du système. Cela comprend aussi la taille des registres stockant les messages échangés entre les processeurs. **[28]**

11. Algorithmique distribué de base

11.1. L'exclusion mutuelle en distribuée

Le système distribué est constitué de **N** processus communiquent entre eux à l'aide d'un système de communication et souvent partagent une ressource logicielle ou matérielle. Un problème d'accès concurrent a lieu quand deux ou plus processus veulent accès (utiliser) à cette ressource. Pour éviter ce problème, la ressource partagée n'est accédée (utilisée) que par un processus à la fois, en d'autres termes, la ressource doit être utilisée en exclusion mutuelle.

Une telle ressource est dite ressource critique et les parties de programme accédant à la ou les ressources partagées et ne devant être exécutées que par un seul processus à la fois sont appelées **sections critiques (**SC). **[01]**

Un processus est dans 3 états possibles, par rapport à l'accès à la ressource:

1- Demandeur : demande à utiliser la ressource, à entrer dans la section;
2- Dedans : dans la section critique, utilise la ressource partagée;
3- Dehors : en dehors de la section et non demandeur d'y entrer.

Changement d'état par un processus:

1- De dehors à demandeur pour demander à accéder à la ressource;

2- De dedans à dehors pour préciser qu'il libère la ressource;
3- Le passage de l'état demandeur à l'état dedans est géré par le système et/ou l'algorithme de gestion d'accès à la ressource. **[32]**

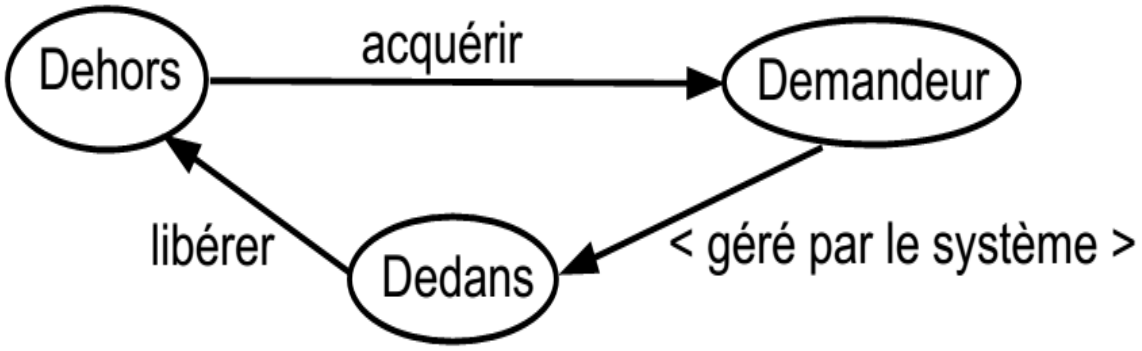


Figure1.6: Diagramme d'états de l'accès en exclusion mutuelle.

Pour garantir l'accès exclusif à la ressource critique, on doit gérer cet accès par un protocole d'acquisition et un protocole de libération. Un processus désirant entrer en SC doit appeler le protocole d'acquisition, ce protocole permet de retarder l'accès de ce processus à la SC si elle est utilisée par un autre site. À la fin de la SC, le processus exécute le protocole de libération pour informer les sites qui peuvent être en attente que la SC est libre. **[28]**

11.2. Transaction distribuée

Une transaction est une unité de travail élémentaire devant s'exécuter totalement ou pas du tout, et dont les effets ne doivent affecter les autres utilisateurs que quand elle est validée. Une transaction peut être décomposée en plusieurs sous-transactions confiées à des sites différents. **[04]**

La cohérence transactionnelle est assurée à travers quatre propriétés, résumées sous le vocable **ACID**:

Atomicité (« All or nothing »): toutes les opérations de la transaction sont exécutées ou aucune ne l'est.

Cohérence (« consistency ») La cohérence signifie que la transaction doit être correcte du point de vue de l'utilisateur

Isolation (pas d'interférence provenant d'autres transactions): une transaction a une opération marquant son début (begin transaction) et une autre indiquant sa fin (end transaction). Si la transaction s'est bien déroulée, la transaction est terminée par une validation (commit). Dans le cas contraire, la transaction est annulée (rollback, abort).

Durabilité (« durability »): une fois que la transaction est validée, ses modifications sont persistantes et ne peuvent être défaites. En cas de panne de système, la durabilité peut être compromise. **[11]**

Les propriétés ACID sont très difficiles à maintenir car elles représentent un frein aux performances du système. Par exemple, Dans un système distribué, il est difficile d'assurer l'atomicité des transactions car plusieurs processus (ou sites) peuvent participer à la réalisation d'une transaction unique et toutes ces sites participant à cette transaction doivent valider localement avant que la transaction ne soit validée globalement.

La défaillance d'un des participants à la transaction ou d'une des liaisons de communication peut aboutir à des résultats erronés. Pour cela, la distribution de calcul impose à chaque site du système de posséder **un coordinateur de transactions local**.

Un coordinateur de transaction est responsable de la coordination et de l'exécution de toutes les transactions effectuées sur son site d'implantation:

- Il démarre l'exécution de la transaction globale;
- Il découpe la transaction en une série de sous-transactions qu'il confit aux sites adéquats;
- Il coordonne la fin de la transaction. Cette terminaison peut aboutir sur la validation ou l'annulation de la transaction globale. Dans le cas d'une annulation, tous les sites ayant participé doivent aussi annuler leur transaction locale. **[04]**

11.3. Diffusion causale

Mode d'émission de messages avec multiples récepteurs.

- Un processus appartient à un groupe global de processus;
- Un message émis par un émetteur est reçu par plusieurs récepteurs;

- Tous les processus du groupe dans le cas de la diffusion (broadcaste), L'émetteur s'envoie aussi le message à lui-même;
- Les processus d'un sous-groupe dans le cas du multicast, Avec possibilité que l'émetteur fasse ou pas partie du groupe des récepteurs et autres variantes;
  - ✓ Ordre et fiabilité de la diffusion des messages.
- Par défaut et selon les temps de propagation, les messages arrivent ... quand ils arrivent;
- Pas dans le même ordre ni dans un ordre donne pour tous les processus;
- Peuvent aussi ne pas arriver chez tous les processus si canaux de communication non fiables ou si l'émetteur plante en cours **[28]**.

11.4. Vote, consensus

Des processus doivent se mettre d'accord sur une valeur commune est y Deux étapes:

- Chaque processus fait une mesure ou un calcul : valeur locale qui est proposée à tous les autres processus;
- Les processus, à partir de toutes les valeurs proposées, doivent se décider sur une valeur unique commune.

Soit un processus initie la phase d'accord

Soit la phase d'accord est lancée à des instants prédéterminés **[28].**

12. Les Problèmes des systèmes distribués

Actuellement, les applications distribuées apparaissent comme une dernière couche au-dessous de laquelle il y a plusieurs autres. On en distingue trois telles qu'illustrées par la figure 1.7.

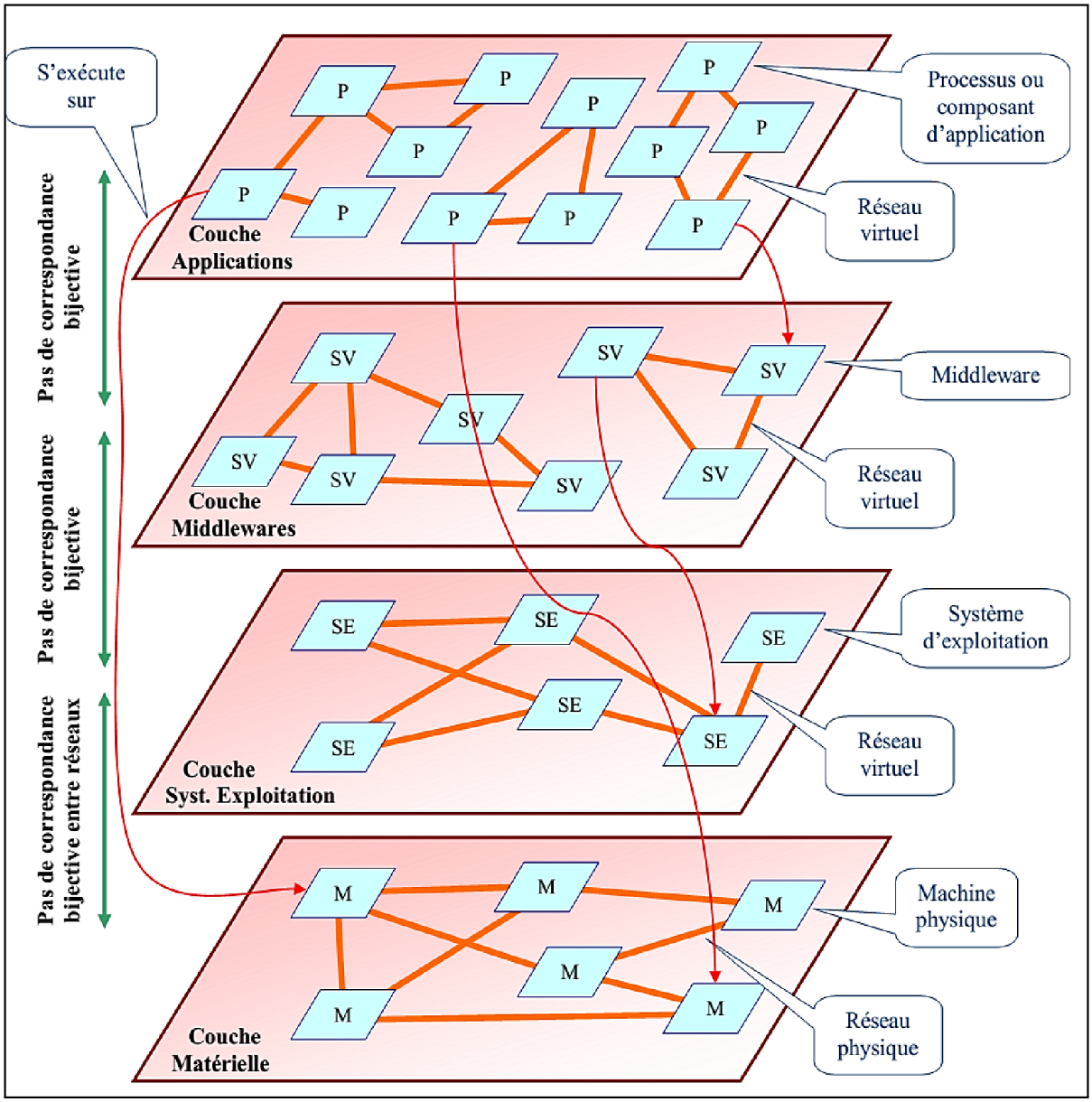


Figure 1.7: Structuration en couches des systèmes distribués.

Dans chaque couche, il est possible d'avoir diverses entités reliées par un réseau qui est physique dans la couche matérielle et virtuel dans les autres couches. Chaque couche matérialise, à elle seule, un système distribué. Cependant, comme les entités de chaque couche s'exécutent en utilisant les services disponibles dans les inférieures, les applications distribuées sont donc mises en œuvre moyennant plusieurs systèmes distribués. La couche middleware est une couche spécifique pour les systèmes distribués. Qu'est censée cacher les subtilités des couches inférieures et offrir aux concepteurs des applications distribuées des services plus convenables de nature à réduire leur complexité **[27]**.

12.1. Problème d'hétérogénéité (Heterogeneity)

Les composants d'un système distribués sont appelés à coopérer pour partager les ressources mais chaque composant peut avoir ses spécificités qui limitent ce partage ou le rendent difficile.

L'hétérogénéité au niveau matériel vient de la différence qui existe entre les ordinateurs aussi bien qu'entre les réseaux. Dans un même système distribué, il est possible de trouver un ordinateur personnel, une station de travail sophistiquée ou même un ordinateur multiprocesseur puissant **[27]**.

12.2. Problème de la concurrence

Le problème de la concurrence se pose pour les systèmes distribués comme pour les systèmes centralisés (i.e. multiprocesseurs). La gestion de la concurrence dans les systèmes distribués nécessite des mécanismes et des outils tout comme les systèmes centralisés (moniteurs, sémaphore). Cependant, dans le cas des systèmes distribués, la gestion de la concurrence se base en partie sur la communication par messages et cette dernière introduit d'autres sources de difficultés, telles que la perte de messages ou leur corruption, d'en-t-il faut tenir compte **[27]**.

12.3. Problème de la sécurité

Les problèmes de sécurité se posent pour tous les systèmes informatiques. Cependant, dans les systèmes distribués, la vulnérabilité se trouve accentuée par la répartition même. Ainsi, il est naturel dans le commerce électronique qu'une partie de l'application (un client) envoie des messages à un autre (un serveur) malheureusement le message peut être intercepté lors de sa transmission, et son contenu divulgué, ce qui peut entrainer de conséquences graves aussi bien pour le client que pour le serveur **[27]**.

12.4. Problème des pannes

Les pannes peuvent apparaitront aussi dans les différentes couches et se propager éventuellement aux autres couches. L'origine des pannes peut être une raison matérielle ou une raison logique liée à la conception des applications, des middlewares et des systèmes d'exploitation. Certaines peuvent avoir comme origine une faille dans gestion de la

concurrence ou la coordination en général. Les pannes d'ordre logiques sont innombrables du fait que les applications distribuées sont par nature même, complexes **[27]**.

12.5. Absence des informations globales

Dans le cas des systèmes distribués, la mémoire commune est inexistante. Chaque processus se trouve dans un environnement différent et son état est une information interne à ce dernier. De même, les informations sur les ressources partagées ne sont pas accessibles en temps réel. De ces faits, la coordination d'un ensemble de processus distribués nécessite le recours à la communication par échange de message. Malheureusement, un message parvient toujours à sa destination avec un certain retard et par conséquent l'information qu'il véhicule ne reflète pas toujours l'état d'un processus ou d'une ressource, car ces derniers peuvent changer d'état durant la transmission du message.

Il est impossible d'avoir une globale instantanée de l'état d'un système distribué et cette situation rend la coordination dans ces derniers un véritable challenge **[27]**.

13. Les avantages et les inconvénients des systèmes distribués

13.1. Les Avantages des systèmes distribués

Comme les technologies de l'Internet et de réseau ont progressé dans la sophistication et la fiabilité, les ingénieurs ont créé une nouvelle manière de gérer les services informatiques: systèmes distribués. Au lieu de centralisation des données et de puissance de calcul dans un seul endroit, puis de l'envoyer aux clients, systèmes distribués répartis les données et les tâches de calcul sur plusieurs nœuds qui fonctionnent à l'unisson. Bien que ce type de système présente de nombreux avantages:

13.1.1. Accélération des calculs: Lorsqu'un calcul peut être décomposé en sous calculs réalisables en même temps « sous-calcul parallélisables », les systèmes distribués permettent la répartition de cette charge sur différents sites. Si un site est en position de surcharge, certains calculs peuvent être déportés sur un site moins chargé. **[04]**

13.1.2. Disponibilité et flexibilité: Un élément peut tomber en panne sans bloquer tout le système et sans influer sur le fonctionnement total du système. **[31]**

13.1.3. Partage de ressources: Lorsque plusieurs sites sont interconnectés, les utilisateurs d'un site ***A*** peuvent utiliser les ressources d'un site ***B*** distantes. [**04]**

13.1.4. Performance: lors d'une transaction entre deux pays, les données utilisées seront principalement celles situées dans les comptoirs des pays concernés. Ainsi, on limite d'autant la congestion d'un système central. En outre, cette dispersion en unités indépendantes et autogérées permet de mettre à jour plus simplement les données ayant attrait, par exemple à la législation locale d'un pays. **[28]**

13.1.5. Fiabilité: un des buts des systèmes répartis est d'obtenir des systèmes plus fiables que les monoprocesseurs. Si une machine tombe en panne, une autre machine devrait prendre la relève, cette situation entraîne généralement la duplication des données partagées pour accroître la robustesse du système. **[33]**

13.1.6. Communication entre les systèmes: Lorsque plusieurs systèmes sont interconnectés par un réseau de communication global, les hôtes de plusieurs sites peuvent échanger des données. Au niveau du réseau, les données sont transmises entre les processus par des flots de messages.

Etant donné cette possibilité de transfert de messages, l'ensemble des fonctionnalités, accessibles au niveau utilisateur local, peut être étendue de manière à englober le système réparti. (Transfert de fichiers complets, navigation sur le WEB, appel de procédures distantes …). [**04]**

13.2. Les inconvénients des systèmes distribués

Les systèmes distribués ne sont pas sans inconvénients. En effet ce sont des problèmes concernant surtout la communication, les avantages offerts par les systèmes répartis nous aident à exploiter le matériel et à concevoir des systèmes informatiques larges et performants, mais ces systèmes ont toujours des problèmes à résoudre:

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

13.2.2. La lenteur de la communication: malgré le degré de fiabilité offert par les réseaux de communication, et les vitesses de transmission qu'ils proposent(avec l'arrivée des fibres optiques), les réseaux restent relativement, et on dit bien relativement lents par rapport à la vitesse de calcul sur les machines, ce qui implique un grand temps d'attente dans le cas d'un travail coopératif entre les processus, et donc on obtient une perte concernant la performance du système.

13.2.3. La perte des messages: parfois, les réseaux de communication ne sont pas fiables, quelques messages transportés par ces réseaux peuvent être perdus, et donc on doit réémettre ces messages, les protocoles de détection, de perte, et la génération des nouveaux messages reste jusqu' à maintenant un des domaines de recherche dans les systèmes répartis. **[33]**

14. Domaines d'application du système distribué

- Ingénierie simultanée
  - Coopération d'équipes pour la conception d'un produit;
  - Production coopérative de documents;
  - Partage cohérent d'information.
- Gestion intégrée des informations d'une entreprise
  - Base de données est divisée en bases de données distinctes coopérant;
  - Les utilisateurs ont accès à des données locales et distantes.
- Contrôle et organisation d'activités en temps réel
  - Fournir des informations précises et opportunes en tout temps;
  - les paiements électroniques;
  - la surveillance du système.
- Centres de documentation,
  - Recherche, navigation, visualisation multimédia. **[12]**

15. Conclusion

Dans ce chapitre, nous avons présenté et formalisé la notion générale de système et algorithme distribué, les différents composants qui le constituent.

Les systèmes distribués ont résolus beaucoup de problèmes, et avec l'évolution de la technologie du monde actuel, le système distribué joue un rôle important dans les différents domaines (scientifiques, commerciales,...). Il est devenu classique de s'appuyer sur multiples unités distribuées pour améliorer la performance d'une application, la tolérance aux pannes, ou pour traiter problèmes dépassant les capacités d'une seule unité de traitement. La conception d'algorithmes adaptés au contexte distribué est particulièrement difficile en raison de l'asynchronisme et du non-déterminisme qui caractérisent ces systèmes. La simulation offre la possibilité d'étudier les performances des applications distribuées sans la complexité et le coût des plates-formes d'exécution réelles.

Les algorithmique distribuées permettent l'utilisation de méthodes non réalisables au sein d'une algorithmique classique. l'objectif de l'algorithmique distribuée est de réunir un groupe d'ordinateurs reliés en réseau afin de former un système de calcul équivalent à une machine vectorielle de très grande puissance et pour l'utilisateur d'un tel système, celui-ci doit être vu comme une seule machine, mono-utilisateur et monoprocesseur, de grande puissance.

Chapitre II

Le problème des pannes en systèmes distribués

1. Introduction

Tous les systèmes informatiques peuvent échouer, les pannes sont partie de tout système. Le véritable problème est la fréquence des échecs et leurs conséquences, ceci relève de la responsabilité des concepteurs de systèmes d'où il est raisonnable de planifier et de prévoir pour chaque conséquence les éventuelles défaillances possibles.

C'est simple dans le cas des systèmes centralisés: si un élément est défectueux (mémoire, processeur, disque-dur, ...) on le change, et on continue ou on recommence le calcul. En distribué, et en l'absence de tout contrôle centralisé, la détection même d'une défaillance peut être difficile. Un lien défectueux peut causer de grave problème (omission, ajout ou corruption de messages). D'autre part, redémarrer tout un protocole sur un système entier peut simplement être infaisable, comme le cas du routage dans Internet. **[13]**

Un trait caractéristique des systèmes distribués qui les distingue des systèmes centralisés est la notion de défaillance partielle: une partie du système est défaillante alors que la partie restante continue à fonctionner, et apparemment correctement. Un objectif important dans la conception de systèmes distribués est de construire le système de manière telle qu'il peut récupérer automatiquement des défaillances partielles sans affecter sérieusement la performance globale. **[06].** En particulier, même avec un (voir plusieurs) lien ou processeur, en panne le programme peut encore fonctionner peut-être avec des performances moindres pendant que les réparations sont en cours. En d'autres termes, le système distribué doit être tolérant aux pannes.

2. Les différents types de pannes

Les pannes dans un système distribué peuvent être de différente nature et affectent divers composants (des processus, des canaux de communication et des serveurs). Une panne peut être franche, transitoire, intermittente ou byzantine.

Type de panne	Description
Panne franche « crash »	Un serveur se bloque, mais il fonctionne correctement jusqu'à ce qu'il s'arrête.
Par omission Omission en émission Omission en réception	Un serveur ne réussit pas à répondre à des demandes entrantes. Un serveur ne réussit pas à envoyer des messages. Un serveur ne réussit pas à recevoir des messages entrants.
Pannes temporelles	La réponse d'un serveur se trouve en dehors d'un intervalle de temps spécifié.
Panne de réponse Panne de la valeur Panne d'Etat-transition	La réponse d'un serveur est incorrecte. La valeur de la réponse est fausse. Le serveur dévie du flux de contrôle correct.
Pannes arbitraires	Un serveur peut produire des réponses arbitraires à des moments arbitraires.

Tableau 2.1: Différents types de défaillances (pannes).

2.1. Panne franche « crash »

Une fois le composant en panne franche il cesse immédiatement et de façon indéfinie de répondre à toute sollicitation ou de générer de nouvelles requêtes (jusqu'à une réparation). Une panne franche est une panne permanente. **[07]**

Un aspect important de défaillances de crash est qu'une fois que le composant est arrêté, rien ne se fait entendre de plus. **[06]**

- **Système stoppé sur panne « Fail-stop »:** un processus s'arrête et reste hors-service. Les autres processus peuvent détecter que ce processus est planté.
- **Système silencieux sur panne « Fail-silent »:** En panne silencieuse le système ne produit plus aucune sortie. Un processus s'arrête et reste hors-service mais les autres processus ne peuvent pas détecter que le processus est planté. **[10]**

2.2. Pannes par omission (transitoire ou intermittente)

Les pannes classées comme des pannes par omission se réfèrent à des cas où un processus ou d'un canal de communication ne parvient pas à effectuer des actions auxquelles elle est supposée faite.

Considérons la communication primitives d'envoyer et de recevoir. Un processus **p** effectue un envoi en insérant le message **m** dans son tampon de message sortant. Le canal de communication transporte **m** au tampon de message entrant **q**. Processus **q** effectue une réception en prenant **m** de son tampon de réception et le livrant (voir Figure 2.1). Les tampons de messages sortants et entrants sont généralement fournis par le système d'exploitation.

Le canal de communication produit une défaillance de l'omission si elle ne transporte pas un message depuis tampon de message sortant de **p** au tampon de message entrant **q**. Ceci est généralement causé par le manque d'espace tampon au niveau du récepteur ou une passerelle intermédiaire, ou par une erreur de transmission de réseau. **[15]**

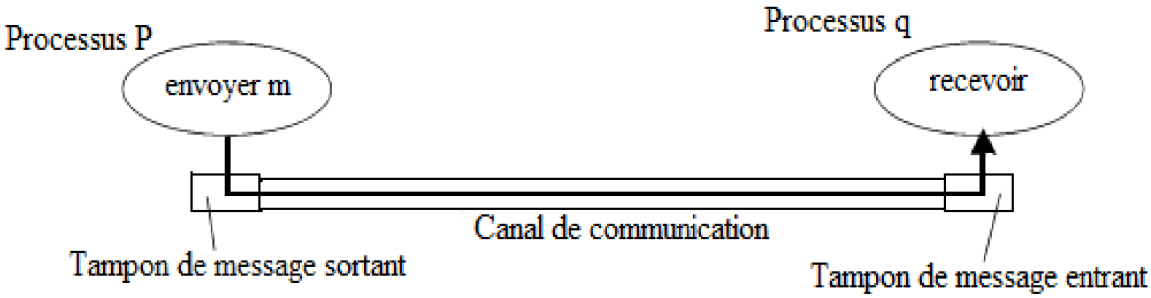


Figure 2.1: Les processus et les canaux de communication.

Hadzilacos et **Toueg** se réfèrent à la perte de messages entre le processus d'envoi et le tampon de message sortant comme des pannes d'omission en émission, à la perte de messages entre le tampon de message entrant et le processus de réception que des pannes d'omission en réception, et à la perte de messages entre eux comme des pannes d'omission du canal. **[08]** En d'autres termes:

- **Omission en émission:** un processus envoie un message mais il n'est pas placé dans le tampon d'émission;
- **Omission en réception:** un message est placé dans le tampon de messages d'entrée d'un processus mais le processus ne le reçoit pas;

- **Omission du canal:** Un message positionné dans un tampon de sortie d'un processus n'arrive jamais dans le tampon d'entrée du destinataire.

2.3. Les pannes temporelles

Les pannes temporelles sont applicables uniquement dans les systèmes distribués synchrones lorsque les délais sont fixés sur le temps d'exécution de processus, le temps de livraison de message et le taux de dérive de l'horloge.

Les pannes temporelles se produisent lorsque: La réponse se trouve en dehors d'un intervalle en temps réel spécifié: dépassement des bornes temporelles, La fourniture de données trop tôt peut facilement causer des ennuis à un destinataire s'il n'y a pas d'espace tampon suffisant pour contenir toutes les données entrantes et le cas le plus fréquent est que le serveur répond trop tard, Dans ce cas on dit une défaillance de la performance est se produire. **[16]**

Les pannes temporelles sont listées dans le tableau 2.2. Chacune de ces défaillances peut aboutir à des réponses étant indisponible pour les clients dans un intervalle de temps spécifié. **[15]**

Classe de défaillance	Affecte	Description
Horloge	Processus	Processus horloge locale dépasse les limites de son taux de dérive du temps réel.
Performance	Processus	Processus dépasse les limites sur l'intervalle entre les deux étapes.
Performance	Canal	Un message émis est reçu après un délai supérieur à la borne prévue.

Tableau 2.2: Les pannes temporelles.

2.4. Panne de réponse

Panne de réponse échec de la valeur - échec Etat-transition

Un type sérieux des pannes est une panne de réponse, par laquelle la réponse du processus est tout simplement incorrecte. Dans le cas d'une panne de valeur, un processus fournit simplement une mauvaise réponse à une demande. Par exemple, un moteur de

recherche qui renvoie automatiquement des pages Web qui ne sont pas liés à l'un des termes de recherche utilisés, il a échoué.

L'autre type de panne de réponse est connu sous l'expression de panne d'état-transition. Ce type de panne se produit lorsque le serveur réagit de façon inattendue à une demande entrante. Par exemple, si un serveur reçoit un message, il ne peut pas reconnaître, une panne d'Etat-transition se produit si aucune mesure n'a été prise pour traiter de tels messages. En particulier, un serveur défaillant peut incorrectement prendre des mesures par défaut, il ne devrait jamais avoir initié.

2.5. Pannes arbitraires (ou byzantines)

Les pannes byzantines sont caractérisées par un comportement déviant du processus affecté: Ce dernier ne respecte plus sa spécification, soit à cause d'une opération malveillante, soit en raison d'une erreur logicielle ou encore à la suite d'une erreur physique non détectée comme une erreur mémoire.[**01**] On distingue deux catégories de pannes arbitraires:

Pannes byzantines "Naturelles": problème non voulu, non déclenché **[10]** Exemple: Erreur physique non détectée (sur une transmission de message, en mémoire, etc...).

Pannes byzantines "Malicieuses »: Erreurs volontaires, comportement visant à faire échouer le système (sabotage, virus, attaques ...). **[07]**

3. Définition de la tolérance aux pannes (Fault tolerance)

Une panne est la manifestation d'un comportement inattendu, et la tolérance aux pannes est un mécanisme qui masque ou restaure le comportement attendu d'un système suivant l'apparition de défaillances (pannes). **[05]**

La tolérance aux pannes est l'ensemble des techniques de conception des systèmes qui continuent de fonctionner même en présence de la panne de l'un de leurs composants. **[07]**

Pour comprendre le rôle de la tolérance aux pannes dans les systèmes distribués, nous avons d'abord besoin d'examiner de plus près ce que cela signifie réellement.

La tolérance aux pannes fait partie d'un domaine plus large appelé la sûreté de fonctionnement (anglais: dependability). La sûreté de fonctionnement est un terme qui couvre un certain nombre d'exigences utiles pour les systèmes distribués, y compris ce qui suit: **[06]**

- Disponibilité (Availability);
- Fiabilité (Reliability);
- Sûreté (Safety);
- Maintenabilité (Maintainability).

La disponibilité ***D(t)***

La disponibilité est définie comme la propriété qu'un système est prêt à être utilisé immédiatement. En général, elle se réfère à la probabilité pour qu'un système soit en fonctionnement correctement à un instant **t** donné **[07]** et il est disponible pour remplir ses fonctions pour le compte de ses utilisateurs. En d'autres termes, un système hautement disponible est celui qui sera le plus susceptible de travailler à un instant donné dans le temps. **[16]**

La fiabilité ***R(t)***

La fiabilité est définie comme la propriété qu'un système peut fonctionner en continu sans défaillance. Contrairement à la disponibilité, la fiabilité est définie en termes d'un intervalle de temps au lieu d'un instant dans le temps. Elle fait référence à la probabilité pour qu'un système soit continûment en fonctionnement sur une période donnée (entre **0** et **t**). **[07]** Un système très fiable est celui qui va très probablement continuer à travailler sans interruption pendant une période relativement longue de temps. Ceci est une différence subtile mais importante par rapport à la disponibilité. Si un système tombe en panne en moyenne pour un, milliseconde apparemment aléatoire toutes les heures, il a une disponibilité de plus de 99,9999 pour cent, mais il est encore peu fiable. De même, un système qui ne plante jamais, mais qui s'arrête pendant deux semaines spécifiques à chaque Août a une grande fiabilité, mais seulement 92 pour cent de disponibilité. Les deux ne sont pas les mêmes. **[06]**

La Sûreté

La sûreté se réfère à la situation que lorsqu'un système échoue temporairement fonctionne correctement, rien de catastrophique n'arrive. En d'autres termes, les échecs temporels n'ont aucune conséquence catastrophique. Par exemple, de nombreux systèmes de contrôle de processus, tels que ceux utilisés pour le contrôle des centrales nucléaires ou d'envoyer des

gens dans l'espace sont tenus de fournir un degré élevé de sûreté. Si de tels systèmes de contrôle échouent temporairement pour seulement un très bref moment, les effets pourraient être désastreux. De nombreux exemples tirés du passé (et probablement beaucoup d'autres encore à venir) montrent combien il est difficile de construire des systèmes de sûreté. **[16]**

La maintenabilité

La maintenabilité se réfère à la facilité d'un système en panne peut être réparé. Un système hautement maintenable peut également montrer un degré élevé de disponibilité, surtout si les échecs peuvent être détectés et réparés automatiquement. **[06]**

4. Techniques permettant d'assurer la tolérance aux pannes

Après la détermination des types de pannes qui peuvent se produire, ce qui est une tâche difficile, nous devons adapter la spécification du problème à être résolu, en présence de telles pannes (défaillances). Comme nous le verrons bientôt, nous pouvons concevoir des systèmes distribués de telle manière qu'ils peuvent même tolérer ces types de défaillances.

Fischer, Lynch et Paterson ont prouvé que dans un système distribué asynchrone, il n'existe pas d'algorithme déterministe qui résolve le problème du consensus lorsqu'un seul processus tombe en panne.

Dans un système distribué asynchrone, les processus peuvent répondre aux messages à des moments arbitraires, donc un processus crashé est indiscernable d'un processus lent. **[15]**

Le problème du consensus peut être formulé comme suit: un système distribué comporte **n** processus **{0, 1, 2, ..., n-1}**. Chaque processus a une valeur initiale dans un domaine commun accord. Le problème est de concevoir un algorithme tel que, malgré l'apparition de défaillances, les processus éventuellement d'accord sur une valeur de décision définitive et irrévocable qui satisfait aux trois conditions suivantes:

- **La résiliation**: Chaque processus non défectueux doit éventuellement décider.
- **La validité**: Si tous les processus non défectueux commencent par la même valeur initiale **v**, alors leur décision finale doit être **v**.
- **L'accord**: La décision finale de tous les processus non défectueux doit être identique.

Exemple: Cinq capteurs indépendants mesurent la température **T** d'un four. Chaque vérification des capteurs Si **T** est supérieure à **1000°C**. Certains capteurs peuvent être défectueux, mais on ne sait pas lesquels sont défectueux. Les capteurs non défectueux doivent être d'accord sur la vérité du prédicat **T> 1000°C** (de sorte que le prochain plan d'action peut être décidé). **[05]**

Nous allons maintenant décrire certaines techniques qui permettent d'assurer la tolérance aux pannes:

4.1. Gestion de la redondance

La tolérance aux pannes dans un système distribué est assurée par la redondance. En effet, la technique clé pour masquer des pannes est d'utiliser la redondance. Cette redondance peut être spatiale (réplication de composants), temporelle (traitements multiples) ou informationnelle (redondance de données, codes, signatures). **[18]**

4.1.1. Les types de la redondance

4.1.1.1. La redondance informationnelle: Avec cette redondance, des bits supplémentaires sont ajoutés pour permettre la récupération à partir de bits tronqués. Pour une donnée soumise à des erreurs (stockage, transmission) il est d'usage de rajouter des bits supplémentaires pour permettre la récupération à partir de bits tronqués, selon un code correcteur d'erreur qui permet de corriger certaines erreurs **[07]**. Par exemple, un code de Hamming peut être ajouté aux données transmises pour récupérer de bruit sur la ligne de transmission. **[16]**

4.1.1.2. La redondance temporelle: Avec la redondance de temps, une action est effectuée, et ensuite, si besoin est, elle est effectuée à nouveau. Les transactions utilisent cette approche. Si une transaction échoue, elle peut être refaite sans aucun dommage. La redondance temporelle est particulièrement utile lorsque les pannes sont transitoires ou intermittentes. **[06]**

4.1.1.3. La redondance spatiale: Un groupe des processus redondants **G** en redondance spatiale est conçu pour tolérer la panne de certains de ses membres.

En dépit de la panne à un certain niveau de certains membres de **G**, le service offert du point de vue global par **G** continue en masquant le niveau de panne visé. **[07]**

Une redondance spatiale peut donc se faire soit en matériel ou en logiciel. Par exemple, des processus supplémentaires peuvent être ajoutés au système afin que si un petit nombre d'entre eux tombent en panne, le système peut encore fonctionner correctement. En d'autres termes, par des processus reproduisant, un degré élevé de tolérance aux pannes peut être réalisé. **[06]**

Exemple: Dans la figure 2.2 ci-dessous, le Crash de processus **s1** est masqué au client et le système fonctionne correctement selon le point de vue du client.

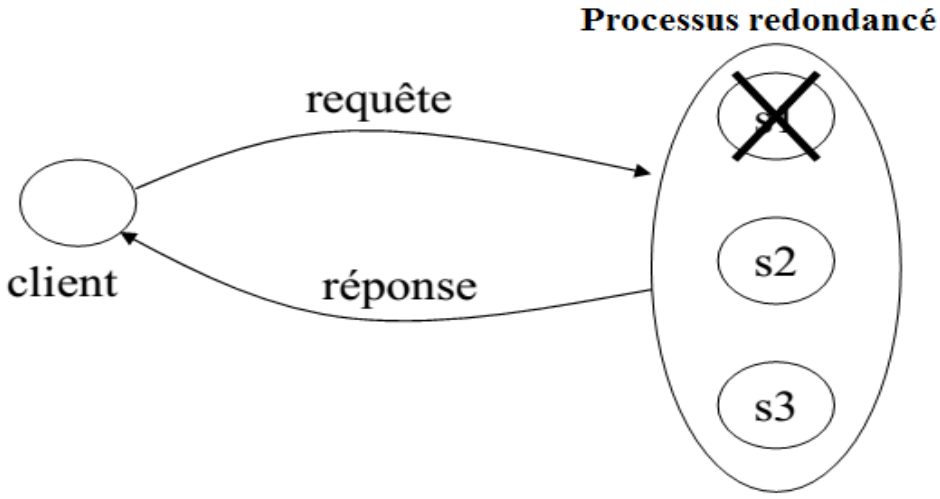


Figure 2.2: La redondance spatiale.

4.1.2. Les mécanismes de redondances

Les mécanismes de redondances mis en œuvre appartiennent à deux catégories : Les mécanismes utilisant la réplication et les mécanismes s'appuyant sur une mémoire stable. Ces mécanismes sont présentés dans les paragraphes suivants **[18]**:

4.1.2.1. Redondance spatiale et temporelle: (Réplication) La tolérance aux fautes par réplication consiste à utiliser des copies multiples d'un même composant ou processus. De cette manière, en cas de défaillance de l'un des composants, la défaillance peut être masquée par l'une des copies. La principale difficulté de cette approche est de conserver une cohérence forte entre les copies. Il existe quatre stratégies principales permettant d'assurer cette cohérence; **[14]**

La réplication passive [19], On distingue la copie primaire et les copies secondaires. La copie primaire est la seule qui reçoit les requêtes et qui effectue toutes les opérations. Pour

assurer la cohérence, la copie primaire diffuse son nouvel état aux copies secondaires après chaque modification. Cet état sert de point de reprise en cas de défaillance.

La réplication active [19] Elle désigne les stratégies dans lesquelles toutes les copies jouent un rôle identique. Toutes les copies reçoivent la même séquence ordonnée de requêtes, qui sont toutes traitées dans le même ordre. Cette stratégie évite d'utiliser des points de reprise couteux. En revanche, elle nécessite un mécanisme de diffusion atomique et requiert que l'exécution des requêtes soit déterministe pour garantir la cohérence.

La réplication semi-active [14] est une amélioration de la réplication active. À la différence de la réplication active, les copies secondaires attendent une notification de la copie primaire avant de traiter la requête. Cette notification comporte les informations nécessaires qui permettent de résoudre le problème de l'indéterminisme du traitement des requêtes.

La réplication coordinateur/cohortes [14] est également une solution hybride entre la réplication active et la réplication passive. La copie primaire est appelée coordinateur et les copies secondaires sont appelées cohortes. Cette méthode est une réplication passive pour laquelle les requêtes sont transférées à toutes les copies pour éviter de les perdre en cas de défaillance.

Le principal désavantage de cette méthode par réplication est qu'elle nécessite de nombreuses ressources : pour tolérer ***p*** défaillances, il est nécessaire d'avoir ***p+1***composants identiques. Cette méthode n'est donc pas adaptée aux calculs parallèles où la performance (temps de calcul) est souvent le critère prépondérant: les ressources doivent être exploitées en priorité pour le calcul. **[18]**

4.2. L'auto-Stabilisation

En cas de panne d'un nœud, la topologie logique sera déconnectée, c'est-à-dire, qu'il y aura des nœuds qui n'ont aucune arête connectée avec la topologie (sauf dans le cas de panne d'une feuille). L'auto stabilisation permet de passer de cet état illégitime à un état légitime où tous les nœuds sont liés (Figure 2.3). **[20]**

- Cette phase ne sera déclenchée qu'après une panne.

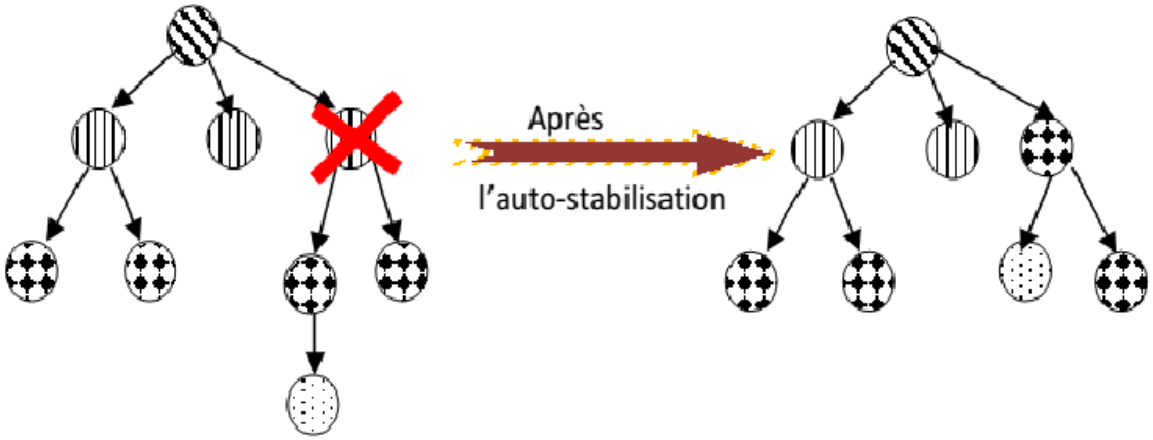


Figure 2.3: L'auto-Stabilisation après panne.

L'auto-stabilisation n'exige pas un état initial et elle est adaptée à la nature dynamique du réseau **[17]**. Comme indiqué, l'auto-stabilisation est la tolérance aux pannes contre les pannes qui se produisent seulement au début et leur durée est limitée.

4.3. Détecteurs de panne

En général, un mécanisme de détection de panne dans un système distribué décrète un délai – time out (retard) – au-delà duquel, si aucune réponse n'a été reçue, le processus est considéré comme étant en panne. Dans un système synchrone pour lequel le délai maximum de communication et de traitement d'une requête est connu, ce système détectera les pannes correctement. Cependant, dans un système totalement asynchrone et pour lequel les délais de réponse ne sont pas prévisibles, comme c'est le cas pour la plupart des systèmes distribués cette méthode ne permettra pas de distinguer un processus en panne d'un processus très lent [**01**].

Détecteurs de panne ont été introduites dans le document séminal de **Chandra** et **Toueg** (Unreliable failure detectors for reliable distributed systems) comme un outil approuvé pour résoudre le problème du consensus. **[21]**

Dans certains détecteurs, la détection des pannes est faite par l'envoi des messages de vie (ce modèle permet de minimiser le nombre des messages nécessaire pour la détection).

Chaque nœud envoie périodiquement un message de vie à son père, si après certain temps ce message n'arrive pas, le père déclare que son fils est nœud en panne et il déclenche la phase d'auto-stabilisation pour garder la connectivité de la topologie. La panne des Cluster-

Head est détectée par la racine. Pour détecter la panne de la racine, elle envoie le message de vie à ses fils.

Il existe deux types de messages de vie selon l'état de l'émetteur. Un nœud peut avoir un des deux états, soit suspect (le degré de vivacité < un seuil) sinon le nœud est considéré comme correcte.

Si le nœud émetteur est correcte le message de vie sera de la forme: <MV1, ID émetteur, IN, OUT > où

- MV1 : indique un message de vie d'un émetteur correcte
- IN : contient la liste des répliques qui ont été ajoutées dans l'émetteur;
- OUT : contient la liste des répliques qui ont été supprimées dans l'émetteur (soit l'utilisateur ou par le système). Généralement IN et OUT contiennent un seul identificateur d'une réplique.

Cette technique permet de contrôler la dynamique des répliques. Si l'émetteur est suspect, le message de vie sera de la forme: < MV2, ID émetteur, numéro de séquence >.

- MV2 : indique un message de vie d'un émetteur suspect.
- Le numéro de séquence: indique le nombre des messages envoyés après la prédiction de panne.

Cette valeur permet de détecter la prédiction incorrecte de panne. Ce type de message permet à un père suspect de contrôler un fils suspect.

On peut avoir trois états :

1. Si le père continuera à recevoir les messages de vie d'un fils suspect, alors après un certain nombre de ces messages, le père déclare que la prédiction choisie sur l'état de son fils est incorrecte;
2. Si les messages de vie d'un fils suspect n'arrivent pas alors le père déclare que ce nœud est dans un état de panne et que la prédiction est correcte;
3. Si les messages de vie d'un fils correct n'arrivent pas alors il est déclaré en panne malgré qu'il n'était pas suspect et les répliques de ces nœuds seront perdus. **[20]**

5. Conclusion

La tolérance aux pannes est un sujet important dans la conception de systèmes distribués. La tolérance aux pannes est définie comme étant la caractéristique par laquelle un système peut masquer la présence et la récupération des pannes. En d'autres termes, un système est tolérant aux pannes si elle peut continuer à fonctionner en présence de défaillances (pannes).

Dans ce chapitre, nous avons présenté les différents types des pannes qui peuvent affecte les systèmes distribués; nous avons aussi mentionné certaines techniques de tolérance aux pannes qui permettent de réagir avant que la panne se présente pour protéger les répliques et augmenter la disponibilité des données, et après l'arrivée de la panne pour protéger la connectivité de la topologie.

Chapitre III

Implémentation et Résultats expérimentaux

1. Introduction

Dans ce chapitre nous allons présenter notre algorithme distribué on utilisant d'un outil qui permet de visualiser et de simuler des algorithmes distribués c'est la plateforme ViSiDiA.

On a proposé une méthode de Gauss distribué pour la résolution des systèmes d'équation linéaire. Et une procédure de faire fonctionner le système de façon acceptable pendant que les réparations sont en cours en cas de panne.

La résolution de systèmes linéaires à grande échelle est un important problème scientifique ou industriel. Ces systèmes linéaires sont souvent révélés sous la forme de matrices non symétriques creuses et de très grande taille. De multiples phénomènes physiques sont modélisés mathématiquement à l'aide d'équations aux dérivées partielles (EDP) qui, après discrétisation, aboutissent souvent à résoudre un système linéaire représenté par l'équation algébrique

$$A\,x = b$$

La résolution de tels systèmes est l'objet de nombreuses recherches aussi bien à propos des méthodes numériques utilisées qu'au sujet de leur réalisation informatique. La taille des problèmes concernés et leur coût en temps de calcul incite à optimiser les méthodes employées et impose l'usage du parallélisme**.**

2. Définition de système d'équations linéaire

Un système de **m** équations à **n** inconnues **X1, X2, ... X_n** s'écrit sous forme matricielle: **AX = B** où **A** est une matrice comportant **m** lignes et **n** colonnes, **X** est la vectrice colonne dont les composantes sont les **xi** et **B**, le second membre, est aussi une vectrice colonne avec **n** composantes. Le vecteur **X** est appelé solution du système **[36]**.

3. Résolution par la méthode de Gauss

Quelles que soient les valeurs **m** et **n** du système, on peut déterminer ses solutions par la méthode d'élimination de Gauss. Le principe en est le suivant : par des combinaisons linéaires successives, on transforme le système initial, que l'on prend tel quel sans changer l'ordre des équations, en un système triangulaire supérieur, système ensuite résolu en commençant par la dernière des équations transformées. On rappelle qu'un système est dit triangulaire supérieur si la matrice associée est triangulaire supérieure. **[36]**

3.1 Algorithme

On peut tirer l'algorithme suivant:

Pour **k** allant de **1** à **n – 1**

Si tous les éléments sous le pivot dans la colonne **k** est nulle, passer à la colonne suivante.

Si l'élément $\mathbf{a_{k,k}}$ est nul, et si il existe un élément non nul, sous le pivot, dans la colonne **k**, permuter la ligne **i** avec la ligne **k** où **i** est le plus petit entier supérieur à **k** tel que , soit non nul. Ensuite, pour tout **i > k**, effectuer l'opération **[37]**.

$$li \leftarrow li - \frac{a_{i,k}}{a_{k,k}} l_k \qquad \textbf{[III.1]}$$

4. Définition de ViSiDiA

ViSiDiA est un logiciel qui existe depuis déjà quelques années au sein du LaBRI, un outil pour prototyper et visualiser les algorithmes distribués.

ViSiDiA permettait de simuler les algorithmes distribués dans le modèle asynchrone avec échanges de messages. **[02]**

4.1. Architecture générale

Nous allons attarder à décrire l'architecture générale de la partie concernant les envois de messages dans ViSiDiA.

Cette partie repose sur trois grandes structures:

- l'interface graphique;
- le simulateur;
- les algorithmes.

4.1.1 L'interface graphique (GUI): L'interface graphique est une partie très importante de ViSiDiA. En effet, elle permet de visualiser en temps réel le déplacement des messages sur le graphe et l'état des sommets. **[25]**

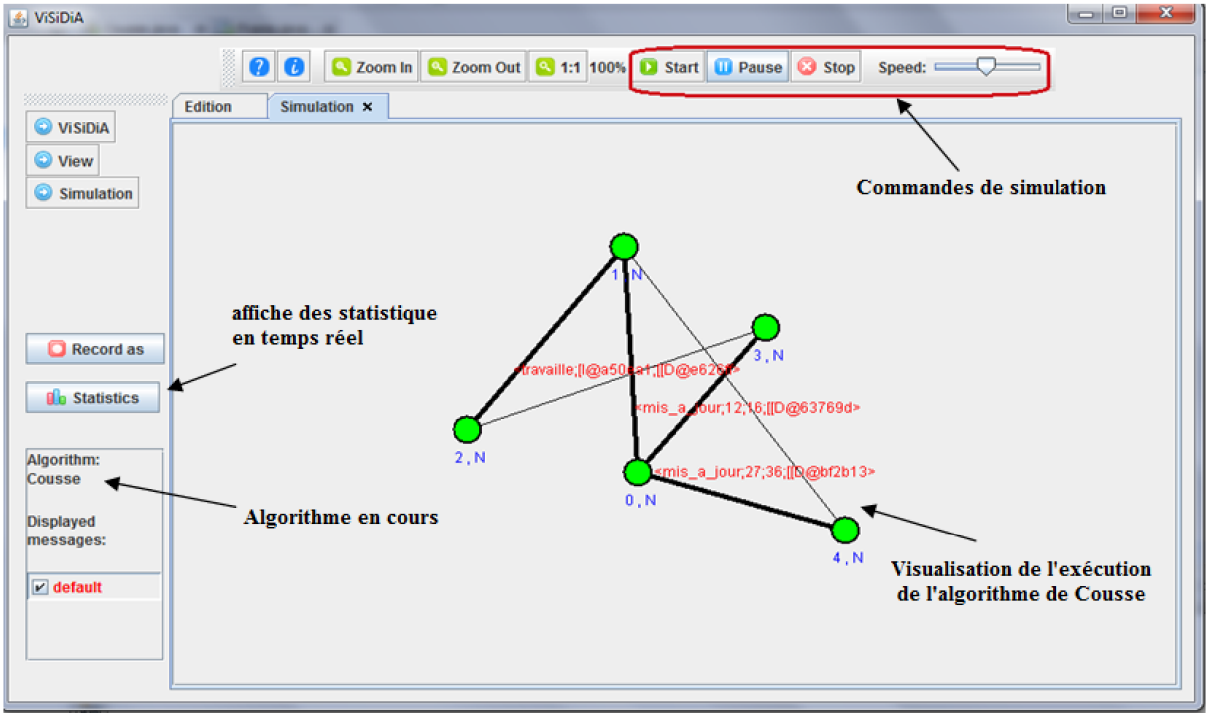


Figure 3.1: Aperçu de l'interface graphique.

4.1.2 Le simulateur: Le simulateur est le lien entre l'interface graphique et les algorithmes. Il modélise un réseau de processeurs asynchrones. Chaque processeur communique seulement avec ses voisins par échange de message. Dans cette version de l'outil, chaque processeur est implémenté par un Thread Java comme indiqué ci-dessus. Le simulateur contrôle les échanges de messages entre les Threads, ainsi que la visualisation des événements. **[35]**

Le simulateur est le noyau de ViSiDiA. Il est entre autres chargé de:

- Lancer l'exécution des algorithmes,
- Gérer les messages du réseau,
- Informer l'interface graphique des changements,
- Valider les acquittements de l'interface graphique,
- Compter les événements en vue d'établir des statistiques. **[25]**

L'interface graphique crée le simulateur et lui fournit le graphe courant, l'algorithme à exécuter choisi par l'utilisateur et deux files serviront à la communication. Le simulateur se charge à ce moment l'a d'initialiser les files de messages pour les algorithmes. Le simulateur crée, pour chaque nœud du graph, un processus qui va exécuter le code de l'algorithme sélectionné. **[25]**

4.1.3 Algorithmes: Les algorithmes représentent la partie évolutive de ViSiDiA. Pour se servir de ViSiDiA, l'utilisateur devra commencer par écrire un algorithme en Java en utilisant l'API fournie. Lors du lancement de l'algorithme, la méthode **init** est appelée. C'est cette méthode que l'utilisateur de ViSiDiA doit implanter. Elle est généralement écrite de la façon suivante:

```
init() {
// Partie d'initialisation
  While (true) {
        // Mécanisme de synchronisation avec un ou plusieurs voisins
        // Envois et réceptions de messages
        // Changement de l'état du sommet et/ou d'une arête
        // Quitter la boucle sous certaines conditions
       }
   }
```

Ainsi, chaque nœud du graphe va exécuter sa propre copie de la méthode **init**. Il faut cependant signaler que toutes les actions que les sommets veulent entreprendre, comme l'envoi de messages, passent par le simulateur. **[25]**

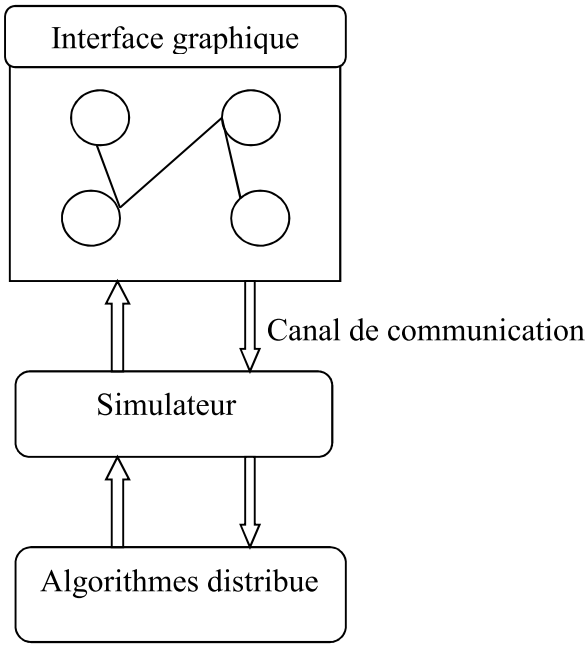


Figure 3.2: Architecture générale de ViSiDiA.

4.1.3.1. La bibliothèque d'algorithmes: Un algorithme est implémenté en Java et sera copié à chaque sommet du graphe et exécuté de manière asynchrone par son processus. Un sommet est implémenté par une classe qui contient son identifiant, son état interne, son degré, et éventuellement la taille du graphe. Il est possible pour le programmeur d'utiliser aussi les primitives suivantes:

- **rendezVous():** une fonction qui retourne le numéro de port du voisin avec qui il y a eu une synchronisation.
- **getId():** retourne l'identifiant d'un nœud (pour les réseaux avec identités),
- **getProperty(String):** (respectivement. putProperty(String,Object)) donne (respectivement. change) d'état d'un nœud,
- **getArity():** retourne le degré d'un nœud, i.e. le nombre de ses voisins,
- **getNetSize():** permet de connaître la taille totale du graphe pour les algorithmes qui ont besoin de cette connaissance.

Puisque les communications entre les processeurs sont basées sur des messages, les fonctions pour manipuler les messages sont fournies. Un message est programmé par une classe qui contient toutes les informations indispensables. Le programmeur peut manipuler un message par les méthodes suivantes:

- **sendTo(voisin, message):** (respectivement. **sendAll(message)**) envoie un message à un voisin particulier (respectivement. tous ses voisins),
- **receiveFrom(int):** (respectivement. **receive(Port)**) reçoit un message d'un voisin particulier (respectivement. le premier message dans la file d'attente du nœud et met dans Port le numéro du port de l'émetteur). D'autres méthodes existent pour manipuler les messages d'un type particulier, i.e. les messages du type entier, chaine de caractères ou vecteurs.

Rappelons que les messages sont stockés dans la file d'attente du récepteur, l'implémentation permet aussi de manipuler les messages qui arrivent sur un canal particulier. **[35]**.

5. Environnement de développement

a. Langage utilisé

Le développement de l'application se fait en langage JAVA, sous l'environnement de développement intégré NetBeans version IDE 8.0.1.

Le langage JAVA possède plusieurs avantages parmi lesquelles que nous intéressent nous citons :

- Utilise la notion de thread ce qui permet de dissocier le calcul des communications.
- Java est portable : il est indépendant de tout plate-forme, il n'y a pas de compilation spécifique pour chaque plateforme.

b. Les ressources

Les expérimentations et les tests ont été exécutés et réalisés sur un « Intel Core i3 » possédant une mémoire de 4Giga avec une vitesse 2.10 GHZ. La plateforme utilisée est Windows 7 et système d'exploitation 64 bits.

c. Paramètre des systèmes linéaires

La matrice **A**: Pour avoir les solutions des systèmes linéaires proposés, on a généré des matrices symétriques définies positives de la forme suivant:

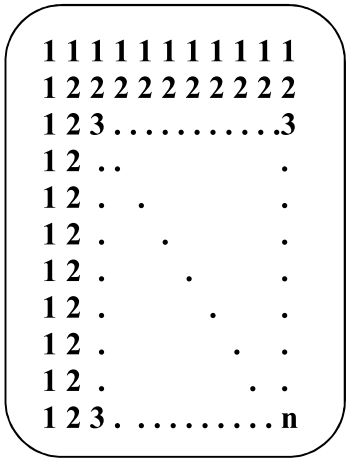


Figure 3.3: Matrices symétriques définies positives.

C'est un vecteur telle que **b(i)** c'est la somme de **i**^{eme} ligne de la matrice **A**. et dans ce cas la solution de système linéaire **Ax = b** sera le vecteur **x= (1,1, ……., 1)**.

6. Implémentation de l'algorithme distribué

Dans la représentation du système distribué exécutant des règles de réécriture, chaque nœud peut être considéré comme une entité autonome de calcul et sera implémenté par un processus, un Thread etc. Chaque arête modélise simplement un canal de communication entre les entités de calculs correspondantes. Le système est asynchrone; i.e. il n'y a aucune horloge globale. Les sommets ont seulement une vue locale du graphe et communiquent seulement avec leurs voisins par des messages asynchrones. En fait, un sommet v est équipé de ports, numérotées de **0** à **(deg(v) −1)**, qui seront utilisés pour communiquer avec ses voisins.

Le fonctionnement général de notre système sera basé sur trois étapes suivant:

- Construire l'arbre sans cycle et couvrant de poids minimal (plus court chemin pour la racine);
- La résolution de système linéaire avec la méthode de Gauss;
- Détection de panne du système et recherche d'une solution.

Ce système respecte les conditions suivantes:

- Si un nœud est tombé en panne alors tous les arêtes de ce nœud sont en pannes et l'inverse correcte;
- On utilise les arêtes qui sont désactivé pour recherche une solution dans le cas de panne;
- Si un partie du système tombe en panne et n'existe aucune solution alors ignoré le partie qui contient pas le nœud racine.

6.1. Construire l'arbre du système

Dans cette phase on a implémenté un algorithme qui permet de construire l'arbre sans cycle et couvrant de poids minimal dans notre système; On a inspiré le principe de notre algorithme de la méthode d'arbre couvrant de poids minimal.

Le nœud (racine) c'est lui qui lancer le système à la première fois.

Nœud racine

Créer un message du type construction de l'arbre sans cycles;

Envoyer ce message à ces voisins;

Fin.

Les autres Nœuds

Recevoir un message du type construction de l'arbre sans cycles;

Faire les deux instructions suivantes une seule fois:

- Activer le porte qui recevoir cette message;
- Envoyer ce message à ces voisins sauf l'émetteur;

Fin.

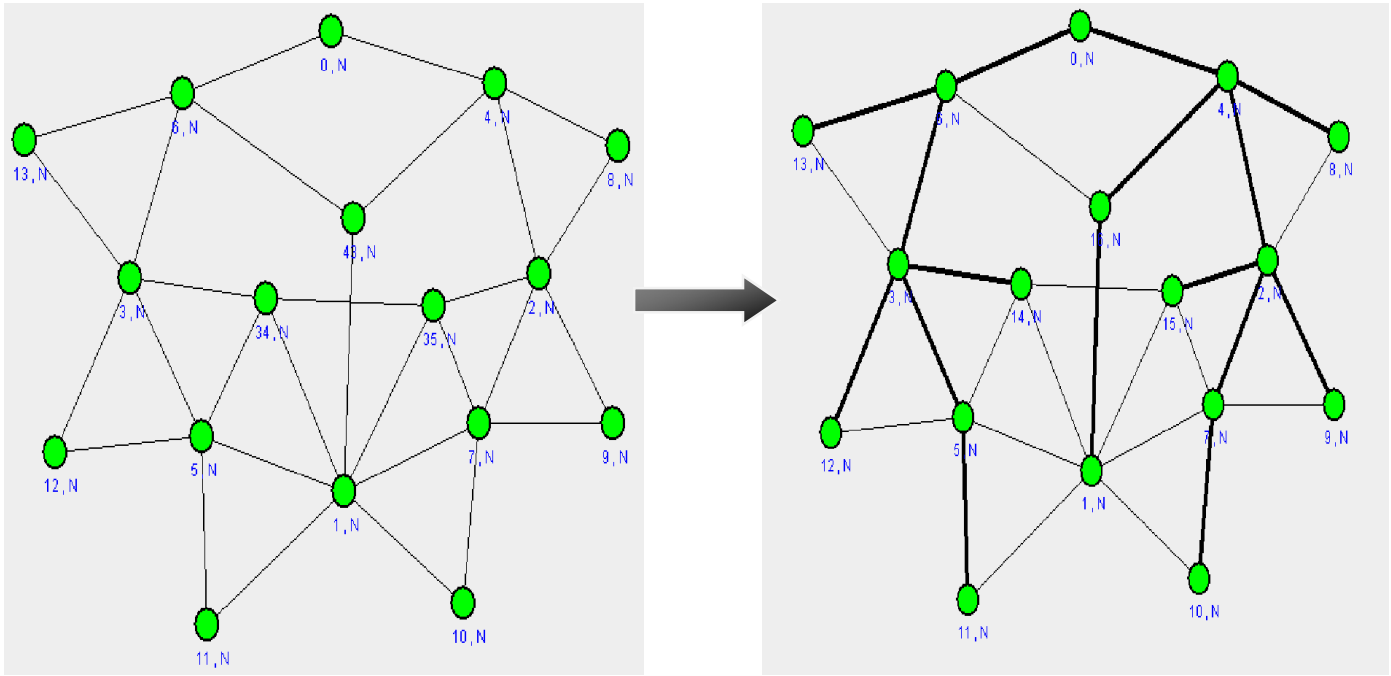
Figure 3.4: Algorithme de construire l'arbre sans cycle.

Cet algorithme capable de donner un graphe sans cycle couvrant de poids minimal (plus court chemin pour la racine) quelle que soit le graph initial.

A l'état initial, on a un graph constitué des nœuds reliés avec des arrêts. La topologie de ce graph peut être connexe et contient des cycles. Pour construire un arbre du système sans cycle; on active et utilise des arrêts tel que ces dernières reliés tous les nœuds sans l'existence d'un cycle entre eux. On désactive les arrêts non utiliser dans l'arbre du système, et on les garde; afin d'utiliser dans le cas du pannes.

Exemple 1:

Dans cet exemple (Figure 3.5) qui Compose de 17 nœuds et plus de 15 cycles, il explique la trace d'exécution de notre algorithme de construire l'arbre sans cycle et couvrant poids minimal.



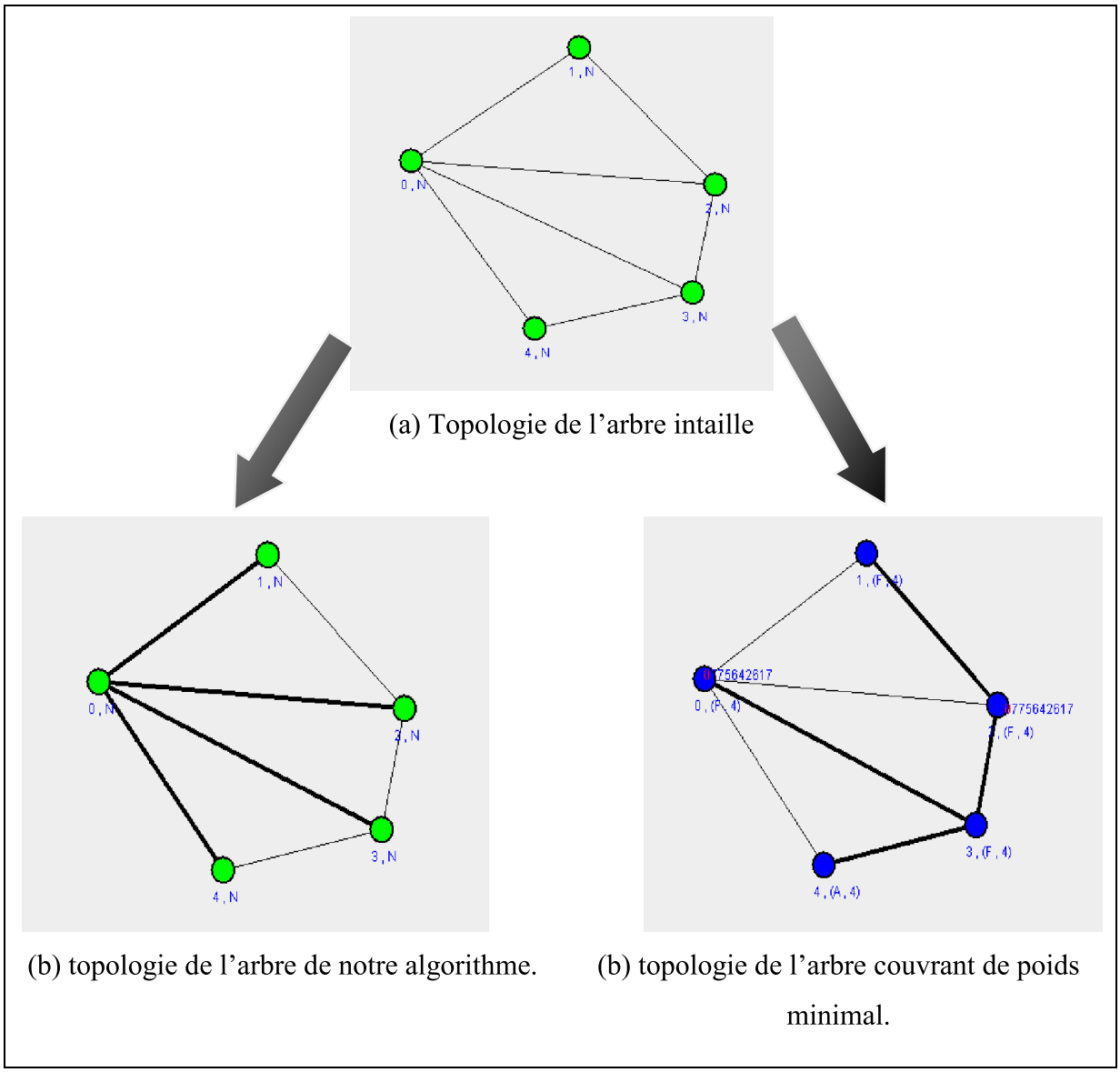
(a) Topologie de l'arbre initiale. (b) Topologie de l'arbre de notre algorithme.

Figure 3.5: Exemple d'exécution l'algorithme de construire l'arbre sans cycle.

La différence entre la méthode d'arbre couvrant de poids minimal et l'algorithme qu'on a proposé est:

- la méthode d'arbre couvrant de poids minimal est construire l'arbre de façon que tous les nœuds soient interconnectés au coût total minimum;
- Dans la situation de notre algorithme, on construire l'arbre de façon qu'elle ne contient pas de cycle et couvrant de poids minimal (plus court chemin pour la racine).

Le plus court chemin nous aide beaucoup dans notre travaille, puisque le nœud racine besoin du travailler avec des autres nœuds et vice versa.



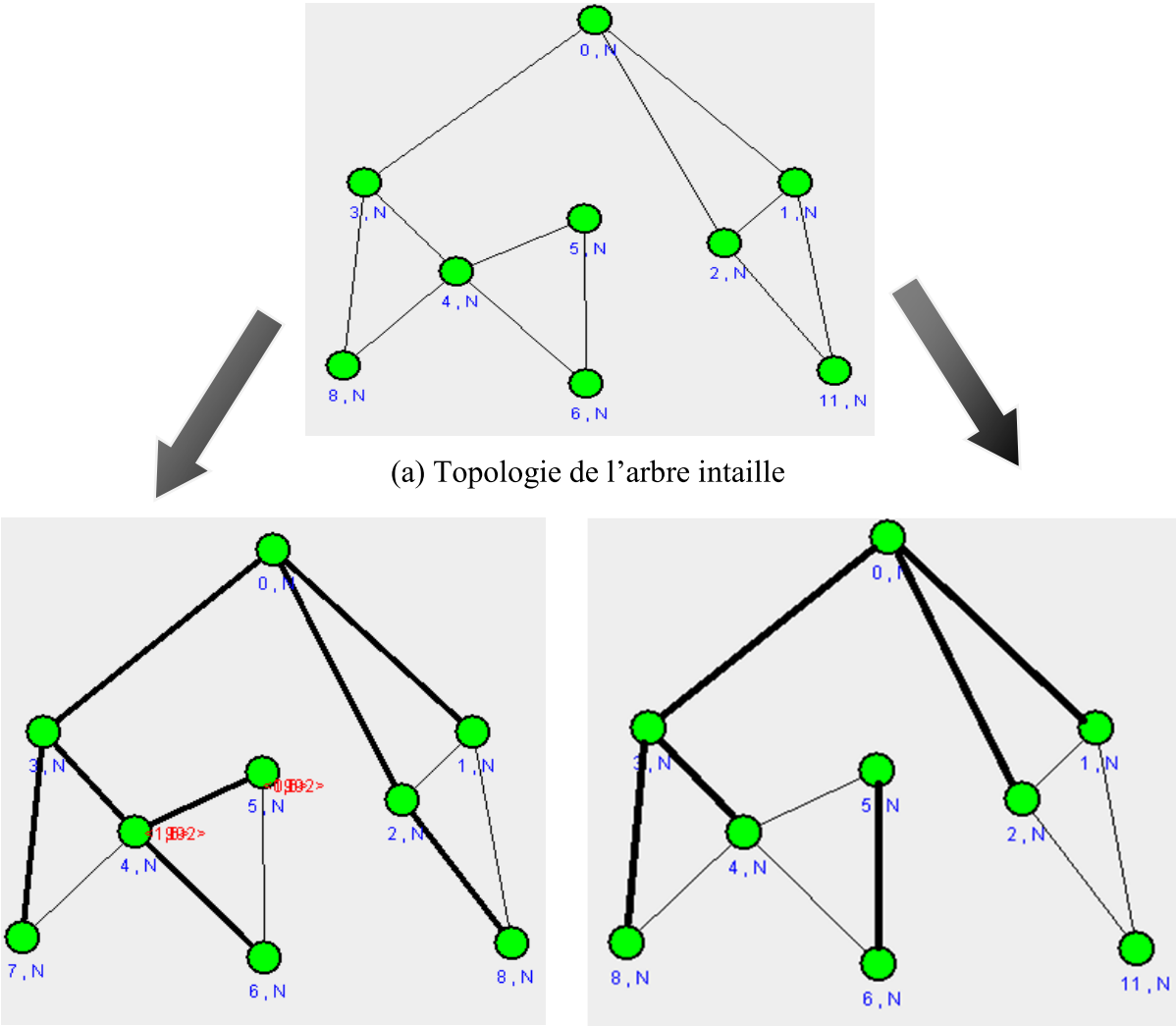
(a) Topologie de l'arbre intaille

(b) topologie de l'arbre de notre algorithme.

(b) topologie de l'arbre couvrant de poids minimal.

Figure 3.6: La différence de construire l'arbre sans cycle entre les deux méthodes.

Avec un exemple simple (Figure 3.7), on remarque que l'algorithme d'arbre couvrant de poids minimal (Kruskal) ne donne pas le résultat souhaité, mais notre algorithme donne un bon résultat (un graphe sans cycle).



(a) Topologie de l'arbre intaille

(b) topologie de l'arbre obtenu par notre algorithme.

(c) topologie de l'arbre couvrant de poids minimal(Kruskal).

Figure 3.7: La comparaison entre les deux méthodes (Kruskal et notre algorithme).

6.2. Architecture de la méthode de résolution proposée

Pour résoudre un système linéaire dense Ax = b par l'élimination (triangulation) de Gauss. On a l'algorithme distribué suivante :

On a choisir d'un nœud (racine) pour charger les paramètres de système linéaire : la matrice A et le vecteur b.

Nœud racine

Construire la matrice augmenter [A,b] ;

Pour **r** de **1 n-1** faire

Diviser et Déterminer la plage des lignes a calculé par la racine et les nœuds voisins ;

Envoyer les messages à ces voisins ;

Calculer les tâches ;

Réception des messages de voisins ;

Rassembler les taches ;

Fin de pour

Calculer la solution.

Les autres Nœuds

Réception le message d'un voisin qu'on l'appelle émetteur;

Diviser et Déterminer la plage des lignes a calculé par la racine et les nœuds voisins sauf l'émetteur ;

Envoyer les messages à ces voisins sauf l'émetteur ;

Calculer les tâches ;

Réception les messages de voisins sauf l'émetteur ;

Envoyer le message au nœud émetteur.

Figure3.8: Algorithme de Gauss distribué.

6.3. Détection de panne du système et recherche d'une solution

Dans cette phase on a implémenté un algorithme qui permet la détection des pannes et rechercher d'une solution s'il existe (au moins une seule solution existe).

Pour modéliser les pannes dans le simulateur ViSiDiA, on a trouvé pas un outil ou bien une technique pour créer une panne dans le système (un nœud tomber en panne ou un arrêt). Pour ce là, on a utilisé un tableau partagée entre les nœuds pour créer les pannes de façon manuelle.

Le principe de Cet algorithme est d'activée la mise à jour de graphe sans cycle qui on a construire à la phase précédente, l'avantage de notre algorithme est qu'elle garde le chemin de graph ancien sauf le sous graphe qui tombe en panne.

- Le Père est le nœud qui deviser le travail en des parties à ce voisins;
- Le Fils est le noud qui recevoir une partie de travaille.

Après la détection de panne en le système

Si le père tombe en panne

- Créer un message de type attendez une solution ;
- Envoyer ce message à tous ces fils;

Fin si.

Si fils tombe en panne

- Désactiver la porte qui recevoir cette message ;
- Créer un message de type recherche d'une solution;
- Envoyer le message à ces voisins sauf le nœud qui est en panne ;

Fin si.

Si réception message de type recherche de solution;

Envoyer ce message à ces voisins sauf l'émetteur ;

Si ce nœud attendez une solution ;

- Activer la porte qui recevoir cette message;
- Créer un message de type demande la dépendance ;
- Envoyer ce message dans la porte qui on a activé ;

Fin si
Fin si

Si réception message de type demande la dépendance ;

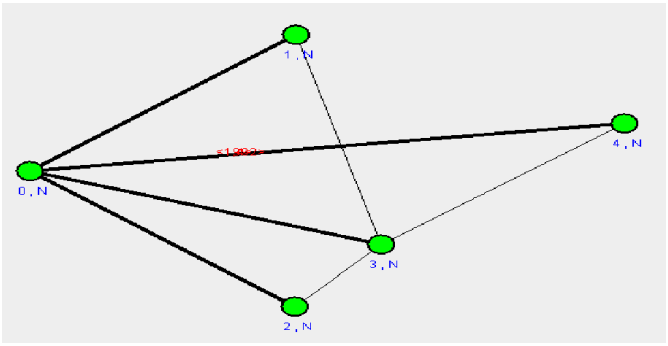
Accepter cette demande ;
Fin si.

Figure 3.9 : Algorithme de trouver une solution de panne.

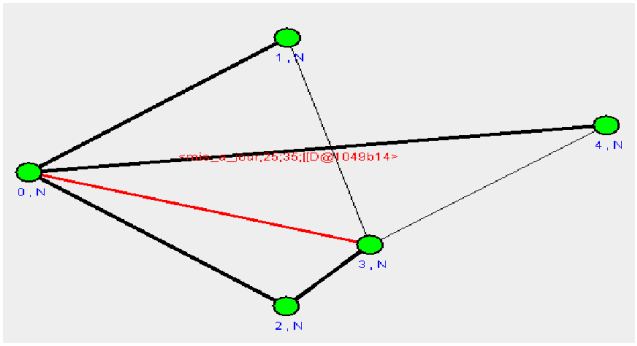
Après le panne de système, on devient un espace du recherche des solutions, pour l'expérience nous obtenons toujours la solution optimale.

Dans les deux exemples suivants (2 et 3), on a expliqué comment l'algorithme obtenue une solution optimale à partir d'un espace de recherche des solutions, dans le cas où l'espace de recherche est vide alors au moins un seul nœud est isolé à l'extérieur du système principal (tomber en panne). On a représenté chaque arrêt qui lié deux nœuds **a** et **b** comme: **arrêt (a-b)** ou **(b-a)**.

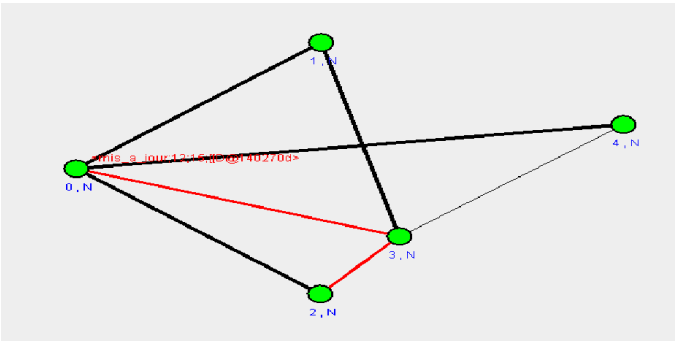
Exemple 2:



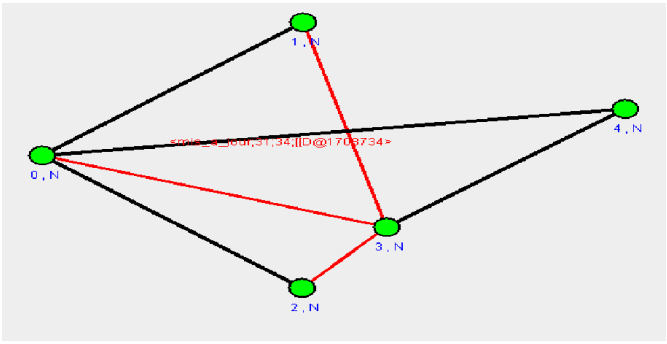
(a) construire l'arbre sans cycle.



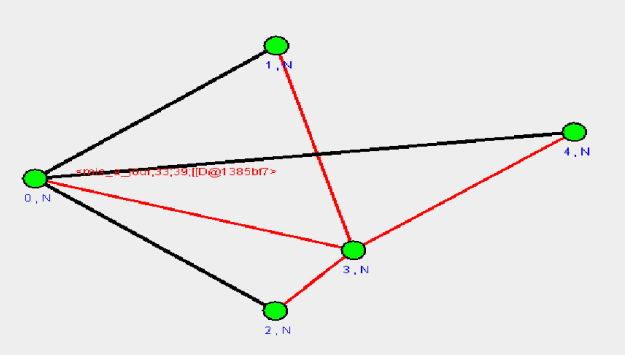
(b) désactiver l'arrêt (0-3), la solution trouver est arrêt (2-3)



(c), désactiver l'arrêt (2-3), la solution trouver est arrêt (1-3)



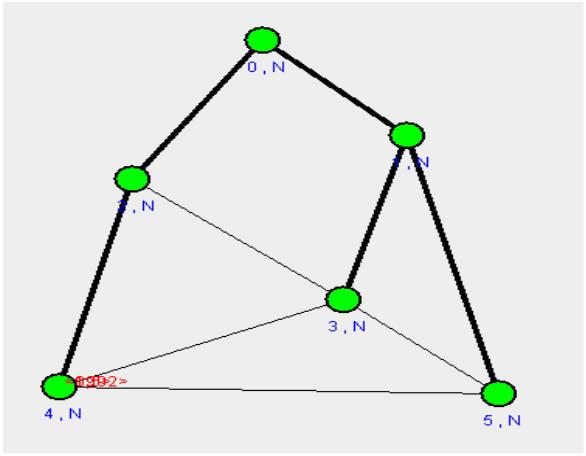
(d), désactiver l'arrêt (1-3), la solution trouver est arrêt (4-3)



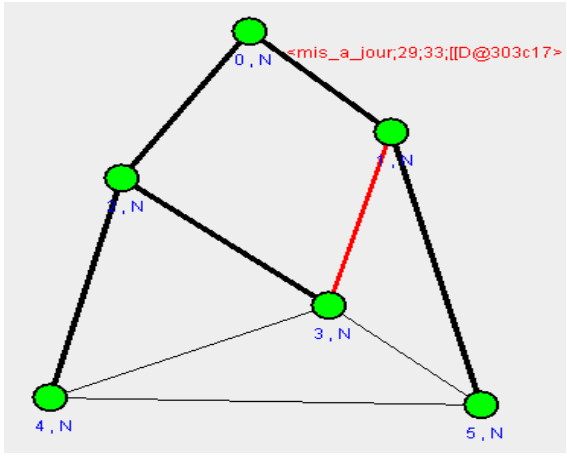
(e), désactiver l'arrêt (4-3) alors nœud 3 est tombe en panne

Figure3.10: Scénario de trouver une solution de panne.

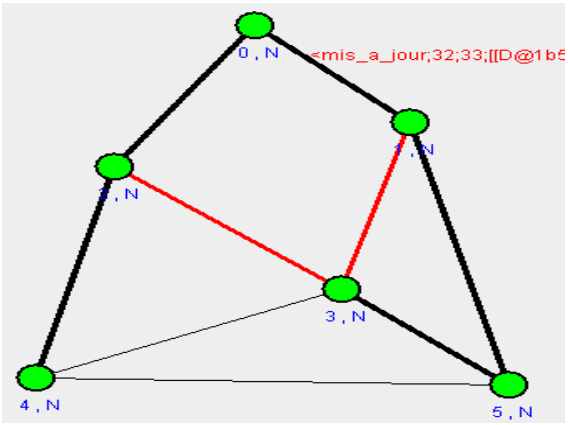
Exemple 3:



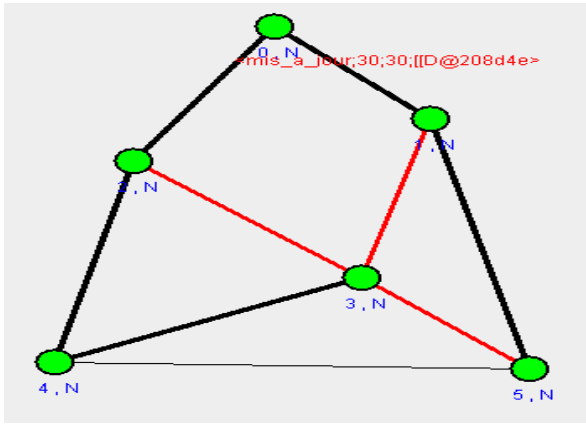
(a), construire l'arbre sans cycle.



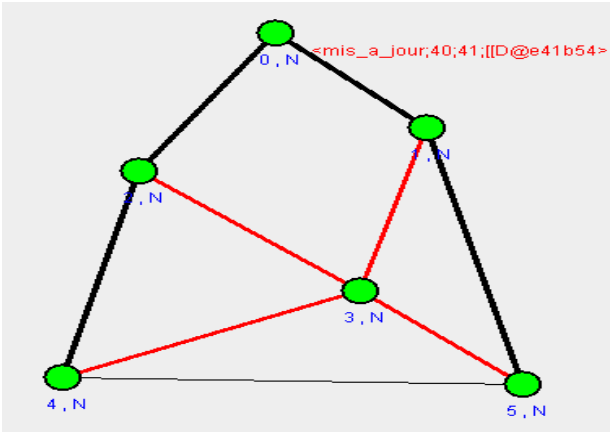
(b), désactiver l'arrêt (1-3), la solution trouver est arrêt (2-3)



(c), désactiver l'arrêt (2-3), la solution trouver est l'arrêt (5-3)



(d), désactiver l'arrêt (5-3), la solution trouver est l'arrêt (4-3)



(e), désactiver l'arrêt (4-3) alors nœud 3 est tombe en panne

Figure3.11: Un autre scénario de trouver une solution de panne.

7. Evaluations expérimentales

Cette étape de test est obtenue en exécutant l'algorithme sans panne et avec panne.

Le tableau résume les mesures obtenues par l'exécution de l'algorithme en variant le nombre de taille de système linéaire pour une topologie content de 22 nœuds.

Taille de système linaire	10	20	30	40	50
Temps total d'exécution (ns)	7422924979	15129250134	29488838385	43039276118	60090093655
Temps total d'exécution avec résolution de panne (ns)	7902598929	20999467237	35986300058	54318984210	74934651880

Tableau 3.1: l'exécution de l'algorithme de 22 nœuds.

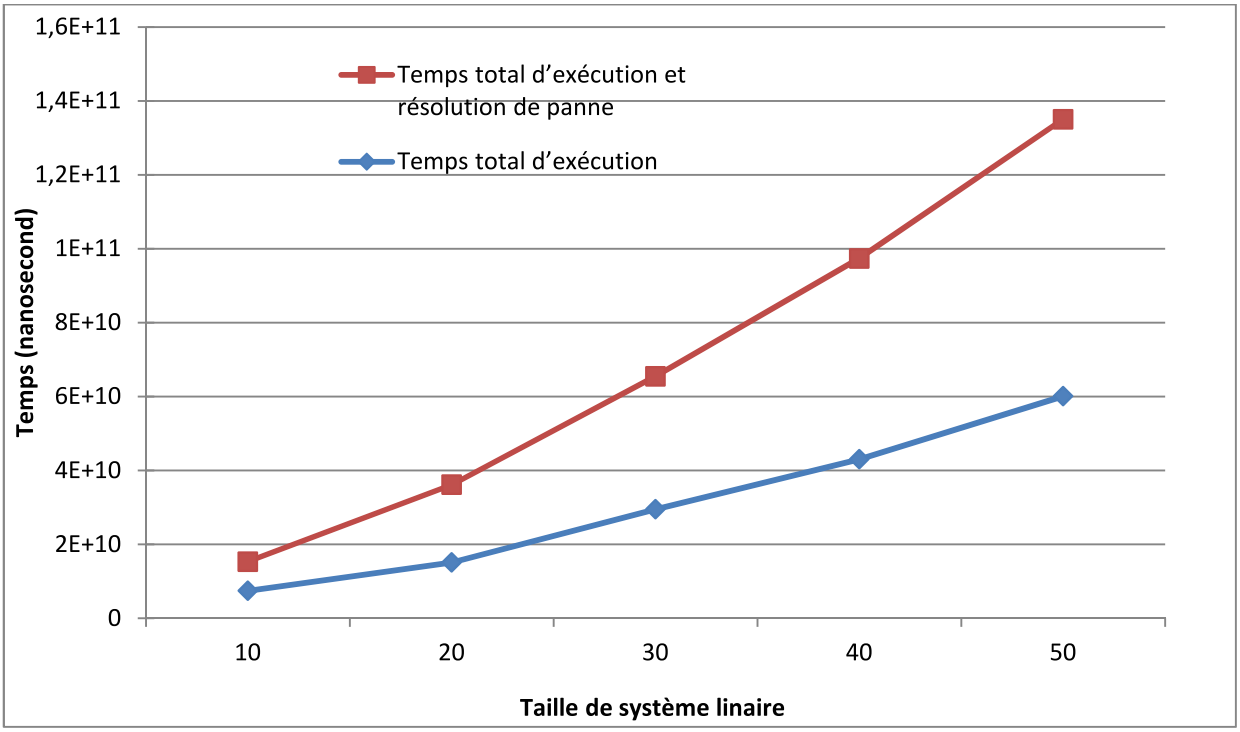


Figure 3.12: Courbe d'exécution de l'algorithme de 22 nœuds.

7.1. Interprétation

On remarque que (voir la figure 3.12):

La différence de temps total d'exécution entre les deux cas, lié linéairement à la dimension de système linéaire. Ceci est dû au l'optimalité de graph et le temps de recherche d'une solution, c'est à dire si le système tombe en panne, Ceci conduit à la diminution de l'optimalité du graph; par conséquent, la perte du temps pour les communications.

8. Exemple d'application

- Premièrement on lance le simulateur ViSiDiA puis on va désunir la topologie que va choisie qui continue les sommets qui relaie par les arrêtes.

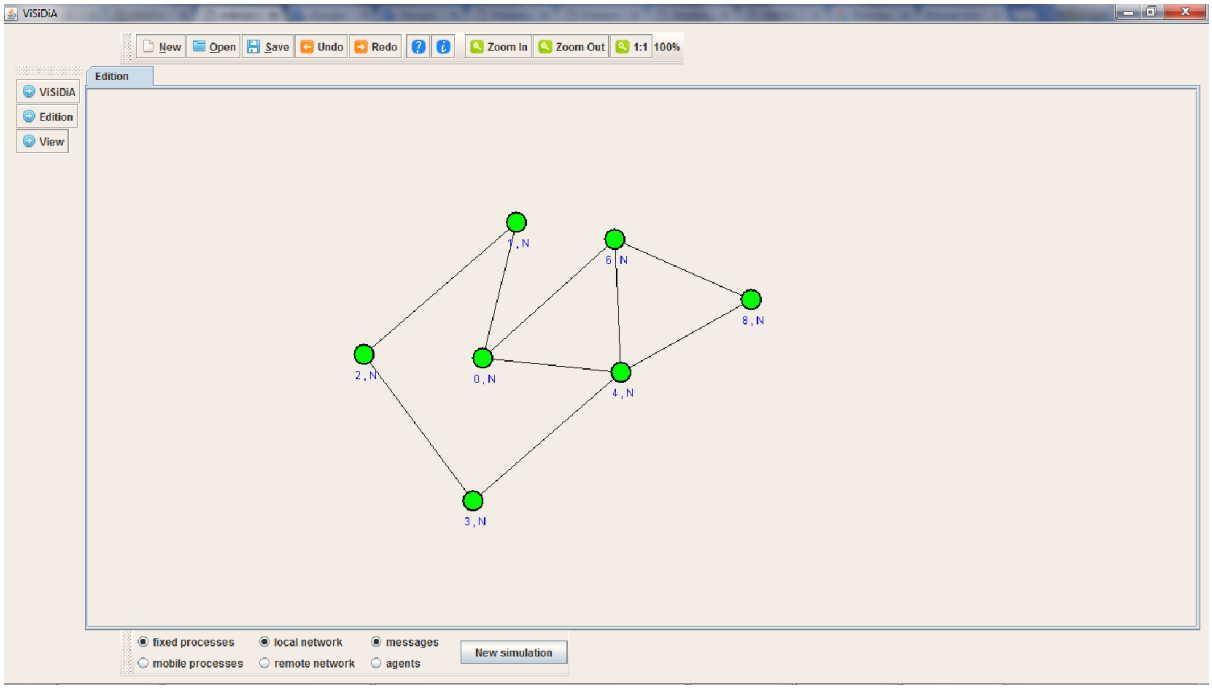


Figure 3.13: Fenêtre exemple d'exécution.

➢ Pour lancer la simulation on cliqué sur le bouton **New simulation.**

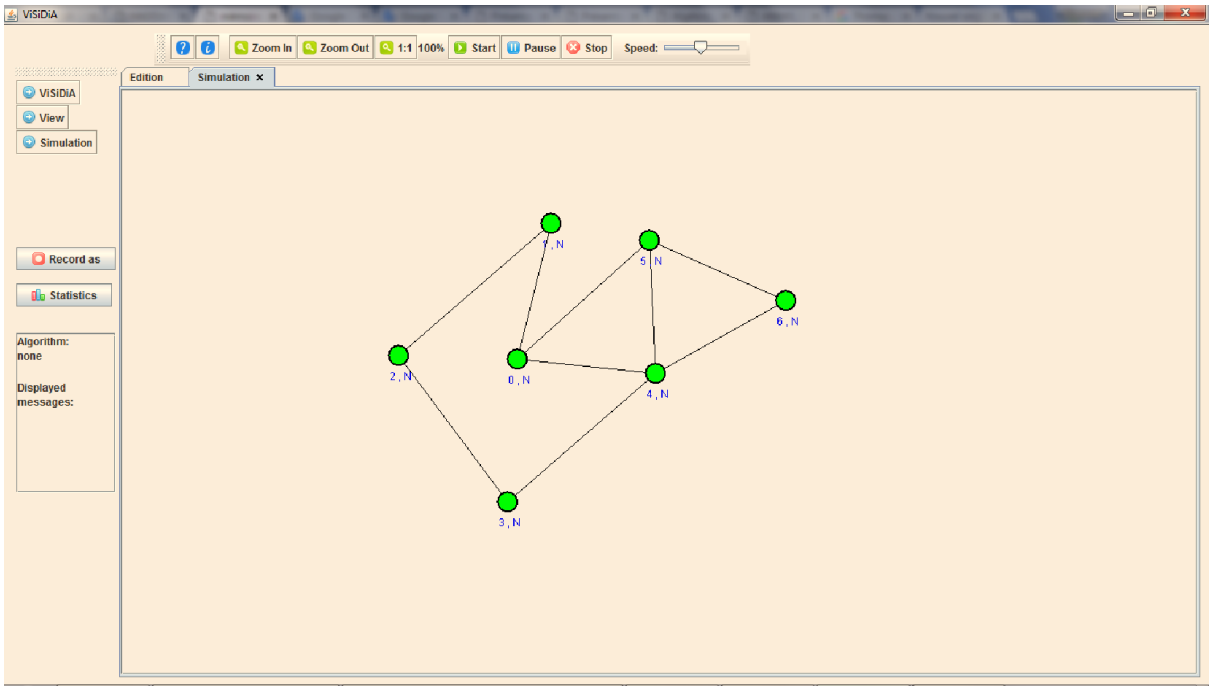


Figure 3.14: Fenêtre exemple d'exécution de simulation.

➢ Ensuite on cliqué sur simulation et cliqué sur select algorithme.

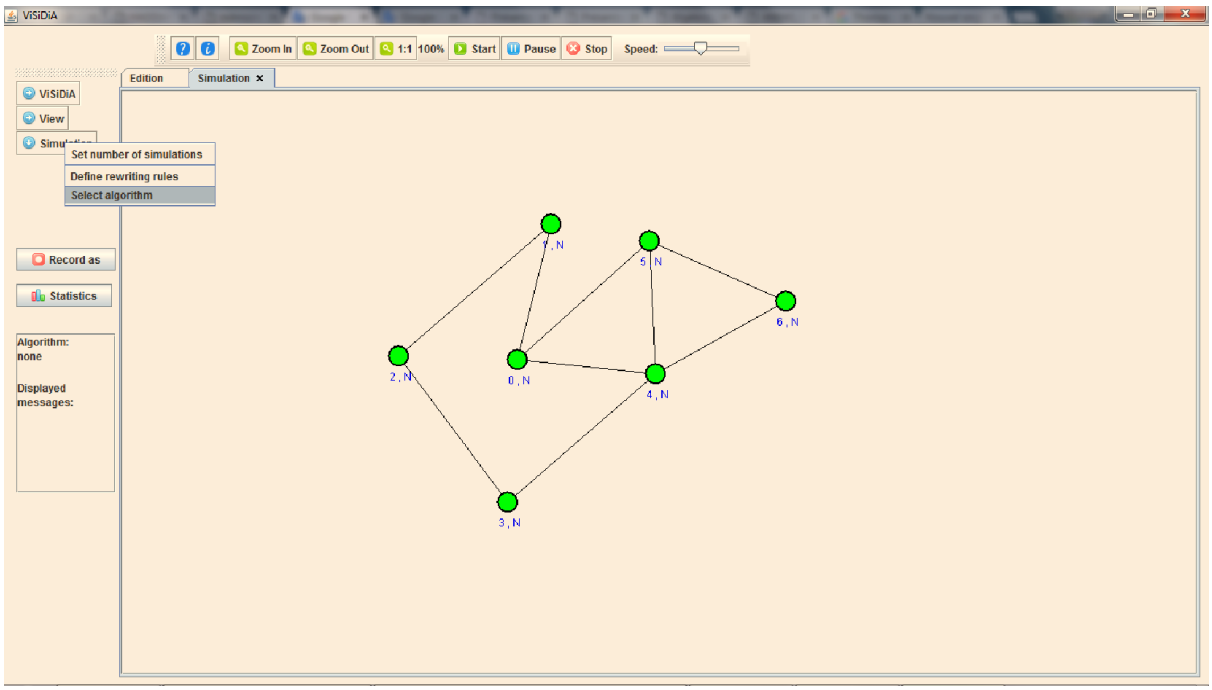


Figure 3.15: Fenêtre exemple d'exécution de simulation de select algorithme.

- Après on cliqué sur **select algorithme** on cliqué sur **User** pour choisir l'algorithme distribué qui exécuter.

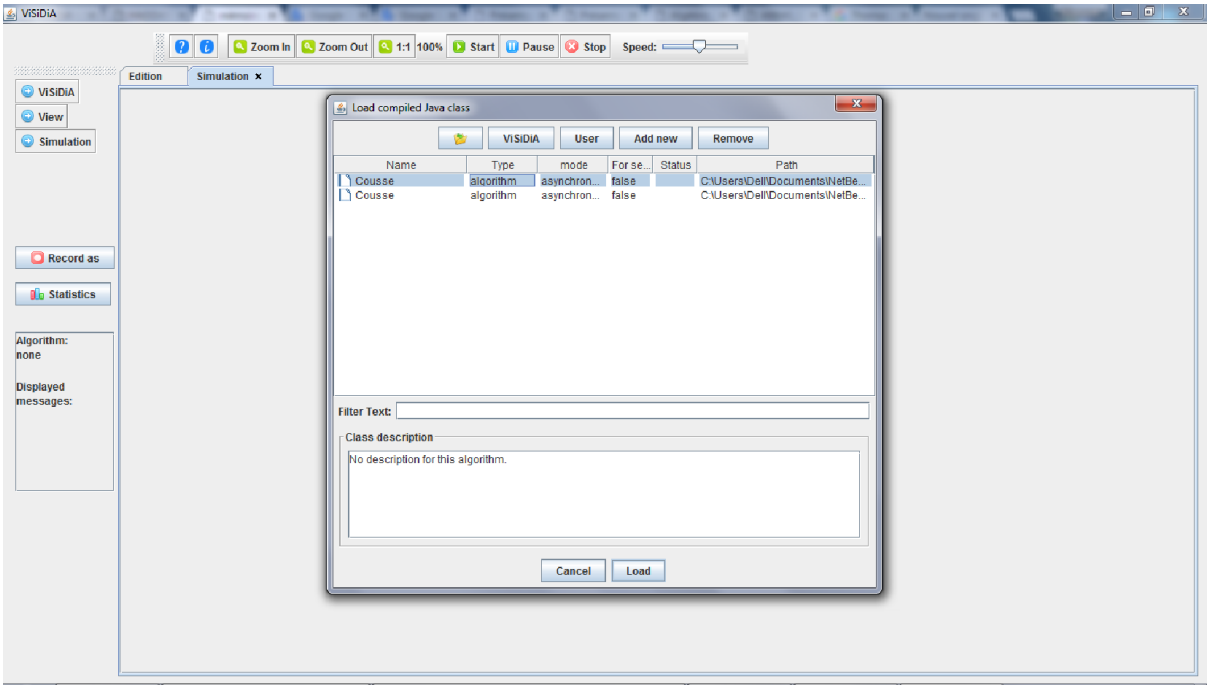


Figure 3.16: Fenêtre exemple d'exécution de simulation choisir algorithme distribué.

- On cliqué sur algorithme qui choisir et cliqué sur **Load.**

Après le lancement de simulation, on a obtenir une fenêtre qui content plusieurs paramètres; On peut choisir le nombre d'équation, le noud racine et on peut choisir aussi l'arrêt que nous voulons d'annuler par le choisir et cliqué sur le bouton **Désactiver**.

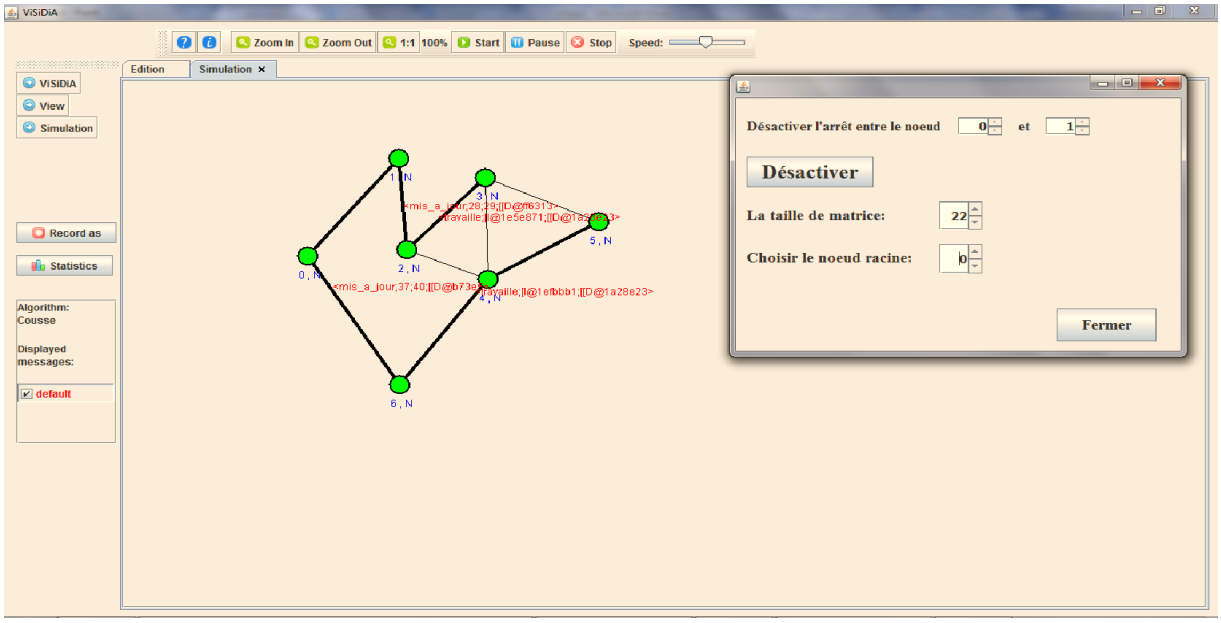


Figure 3.17: Fenêtre exemple d'exécution de notre algorithme distribué en ViSiDiA.

9. Conclusion

Dans ce chapitre nous nous sommes intéressés à la distribution et à l'évaluation de performances du parallélisation des algorithmes d'arbre couvrant de poids minimal et obtenue une solution de panne pour des systèmes linéaires à grande échelle.

Pour cette méthode, l'optimisation de graphe n'influence que la durée totale d'exécution ainsi que la durée de communication augmentent.

On a utilisé la version distribuée de ViSiDiA pour effectuer des expérimentations sur les systèmes linéaires de grande taille. Il serait intéressant de voir comment s'extrapolent ces résultats pour un système réel.

Conclusion générale

Dans ce mémoire on a présenté un état de l'art des systèmes distribués, qui a pour objectif d'utiliser et implémenté un algorithme distribué pour la résolution d'un système d'équation linéaire avec la méthode gauss. Notre algorithme se compose de deux étapes principales; la première étape on a d'abord mise en place un algorithme qui permet de construire l'arbre sans cycle et couvrant de poids minimal dans notre système. Ensuite pour la deuxième étape, on a implémenté un mécanisme qui permettre de détecter des pannes, de recherche d'une solution pour résolu ces pannes, et d'assurer la tolérance aux pannes, cela veut dire un mécanisme qui permettre à le système de fonctionner même en présence de la panne de l'un de leurs composants.

Après l'implémentation et l'expérimenté de notre algorithme dans un nombre énormes des tests (différents topologies du graphe; différents situations des pannes, etc.); on a trouvé que notre algorithme capable de donner un graph sans cycle couvrant de poids minimal (plus court chemin pour la racine) quelle que soit le graphe initiale. Plus de ça, notre algorithme permettre de détecter des pannes et de rechercher puis donner une solution s'il existe (Indépendamment de la taille de l'espace du recherche, s'il existe au moins une solution le système va donner).

Les systèmes distribués ont résolus beaucoup de problèmes, et avec l'évolution de la technologie du monde actuel, le système distribué joue un rôle important dans les différents domaines (scientifiques, commerciales, ...). Mais il y a plusieurs problèmes liés aux systèmes distribués par exemple, dans le cas où un panne affecte tous les arrêts (chemins) qui liés un nœud ou un sous-système avec le système principal (le système qui contient la racine) à cause de ça on obtenue deux systèmes distinctes. Mais dans cette situation on continue le travail seulement avec le système principal et on abandonne l'autre système parce qu'il n'y a aucun chemin qui le reliant avec le système principal. En conséquence, ce problème conduit à une diminution des performances.

Un autre problème c'est parce qu'on a utilisé le mécanisme de détection des pannes alors quand un panne affecte un nœud ou un arrêt, on le détecter puis on résoudre ce panne; Le problème que le nœud qui affecté par ce panne peut être à cette

moment en état de faire un certain calcul ou traitement, et à cause de ce panne ces calculs ou traitements va annuler et perdus, cela coûterait le système du temps et donc une diminution de performance du système. Aussi, dans le cas où l'arrêt qui affecté par ce panne peut être à cette moment en état de transmettre d'informations ou des résultats importants et à cause de ce panne ces informations va perdus, cela coûterait également le système du temps et donc une diminution de performance du système.

Le travail présenté dans ce mémoire peut avoir un impact sur la suite des travaux de recherche à entreprendre dans l'avenir.

On peut citer notamment comme perspectives:

- Dans le cas où le système est divisé en sous-systèmes à cause des pannes; ces sous-systèmes doivent continuer de fonctionnement est donne le résultat;
- pour minimiser le temps d'exécution afin de garder la performance du système, on utilise le mécanisme de prédiction, Cette mécanisme permet de prédire le lieu (nœud ou arrêt), dans lequel il pourrait tomber en panne.

Références bibliographiques

[01] Pascal Grange, Systèmes distribués: transparence, masquage et outils associés, l'université bordeaux I école doctorale de mathématiques et d'informatique, 02 décembre 2005.

[02] Mme Imen Ammari Barouni, Architectures Evoluées, Université de TUNIS, mars 2007.

[03] Rachid GUERRAOUI, Esther PACITTI, Routage des Transactions dans les Bases de Données à Large Echelle, thèse de doctorat, l'Université Pierre et Marie Curie (Paris VI), 2010.

[04] L. Junon, les systèmes distribués, 2008.

[05] Sukumar Ghosh, Distributed Systems: An Algorithmic Approach, Second Edition (Chapman & Hall/CRC Computer and Information Science Series) 2nd Edition, 2007.

[06] Maarten van Steen, Distributed Systems, 2012.

[07] G. Florin, la tolérance aux pannes dans les systèmes répartis, Laboratoire CEDRIC, 24 mars 2013.

[08] Hadzilacos, V. and Toueg, S. A Modular Approach to Fault-tolerant Broadcasts and Related Problems, Technical report, Dept. of Computer Science, University of Toronto, 1994.

[09] Sudheer R Mantena , Transparency in Distributed Systems, 2015.

[10] Eric Cariou, Modélisation conceptuelle des Systèmes Distribués, Université de Pau et des Pays de l'Adour Département Informatique, 2011.

[11] Idrissa SARR, Routage des Transactions dans les Bases de Données à Large Echelle, thèse pour obtenir le grade de docteur, Université pierre et marie curie mention informatique, 07 octobre 2010.

[12] Mohamed Mosbah, Modèles et Approches Formels de Systèmes Distribués, Université de Bordeaux 1, 2006.

[13] Cyril Gavoille, Algorithmes distribués, Laboratoire Bordelais de Recherche en Informatique (LaBRI), Université de Bordeaux, 19 octobre 2015.

[14] Stéphane Drapeau, Un Canevas Adaptable de Services de Duplication, Thèse de doctorat, Institut National Polytechnique de Grenoble - INPG, juin 2003.

[15] George F. Coulouris, Jean Dollimore, and Tim Kindberg, Gordon Blair,Distributed systems (5th ed.): concepts and design, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2012.

[16] Andrew S. Tanenbaum and Maarten Van Steen, Distributed Systems (2nd ed.): Principles and Paradigms, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

[17] F. Reichenbach, Service SNMP de détection de faute pour des systèmes répartis, Ecole polytechnique de LAUSANE, Fevrier 2002.

[18] Thierry Gautier et Denis Trystram, Tolérance aux fautes et reconfiguration dynamique pour les applications distribuées à grande échelle, thèse pour obtenir le grade de Docteur de l'Université de Grenoble, 28 avril 2010.

[19] Matthias Wiesmann, Fernando Pedone et André Schiper, A systematic classification of replicated database protocols based on atomic broadcast, In ERSADS '99: 3rd Europeean Research Seminar on Advances in Distributed Systems, 1999.

[20] Ghalem Belalem, Bakhta Meroufel, Nadia Hadi, Tolérance aux pannes dans les grilles de calcul, Université d'Oran (Es Sénia), 2010.

[21] Brahim HAMID, Distributed Fault-Tolerance Techniques for Local Computations, pour obtenir le grade de docteur, université bordeaux I, 14 Juin 2007.

[22] G. Tel, Introduction to Distributed Algortihms, Cambridge Second Edition, 2000.

[23] Estelle Colin, Thomas Peclier, Fabrice Berna, Algorithme distribue projet: coloration de graphe, 2002.

[24] Shehla ABBAS, Distributed Calculations using Mobile Agents, thèse pour obtenir le grade de docteur, Université de Bordeaux I, 15 décembre 2008.

[**25**] Mohamed amine haddar, codage d'algorithmes distribués d'agents mobiles à l'aide de calculs locaux, l'université bordeaux I, 20 décembre 2011.

[26] Mario Valencia-Pabon, Introduction et modèle distribué, Université Paris-Nord, 25-Sep-2012.

[27] Dr. Djamel Meslati, Systèmes distribués : Principes et concepts, 2010.

[28] Koujouji Djemaa, Nadjemi Zohra, Conception d'un programme distribué pour la résolution d'un système d'équation linéaire, Mémoire de fin d'étude, en vue de l'obtention du diplôme de Master en informatique, Université Ahmed Draia – Adrar, 28/09/2015.

[29] Allyx Fontaine, Analyses et Preuves Formelles d'Algorithmes Distribués Probabilistes, thèse pour obtenir le grade de docteur, Université de Bordeaux I, 16 juin 2014.

[30] Jean-Pierre ELLOY, Bernard ESPIAU, Sur l'optimisation des systèmes distribués réel embarqués, l'université de ROUEN, 2000.

[31] Brice Goglin, Systèmes parallèles et distribués, 2011.

[32] Mr. Mehrez Boulares, Mr. Nour Ben Yahia, Systèmes Répartis, 2013.

[33] Sami et Abdelmadjid Oubbati et Benarfa, **l**a tolérance aux pannes des algorithmes de partage de ressources dans les systèmes répartis et les réseaux **ad hoc (s**imulation par **ns-**2**),** Université Amar Telidji Laghouat - Ingénieur d'état en informatique, 2010.

[34] Eric Cariou, Introduction aux Systèmes Distribués, Université de Pau et des Pays de l'Adour Département Informatique, 2011.

[35] François Morain & Jean-Marc Steyaert, Un environnement général pour étudier et implémenter les algorithmes distribués, 2005/2006.

[36] Escpi-Cnam, Résolution des systèmes linéaires, Février 2005.

[37] Frédéric de Gournay & Aude Ronde pierre, Méthodes directes de résolution du système linéaire Ax = B, 2010.