Université des Sciences et de la Technologie d'Oran
Mohamed Boudiaf (USTO-MB)
Faculté de Mathématique et Informatique
Département d'Informatique

Polycopié du Cours du Module

Algorithmes des Systèmes Répartis

Pour les 1ère année Master
Toutes options confondues

Dr Sarah Benziane

USTO - Université des Sciences et de la Technologie d'Oran
Mohamed Boudiaf

2024

Table des matières

Introduction Générale **1**

1 Généralités sur les systèmes répartis **3**
1.1 Introduction aux systèmes répartis 4
1.1.1 Définition et concepts clés 4
1.1.2 Avantages et inconvénients 4
1.2 Modèles de répartition 5
1.2.1 Modèles architecturaux 5
1.2.2 Modèles de communication 5
1.3 Scénarios de déploiement 6
1.3.1 Cloud computing 7
1.3.2 Internet des objets (IoT) 7
1.4 Évaluation des performances 8
1.4.1 Critères d'évaluation 8
1.4.2 Outils de mesure 8
1.5 Vérification des algorithmes répartis 10
1.5.1 Méthodes formelles 10
1.5.2 Vérification de la cohérence 10
1.6 Conclusion 10

2 La communication **13**
2.1 Introduction 13
2.1.1 Applications des systèmes répartis 13
2.2 Fondements de la communication 13
2.2.1 Modèles de communication 14
2.2.2 Protocoles de communication 14
2.2.3 Mécanismes de communication 14
2.3 Contrôle de flux dans les systèmes répartis 14
2.3.1 Notions de base 15
2.3.2 Stratégies de contrôle de flux 15
2.3.3 Implémentation et exemples 17
2.4 Communication synchrone avec RdV 17
2.4.1 Caractéristiques de la communication synchrone 18
2.4.2 Rendez-vous et synchronisation 18
2.4.3 Implémentation de la communication synchrone 18
2.4.4 Applications et cas d'utilisation 19
2.5 Qualité de service : réseau FIFO 19
2.5.1 Notions de qualité de service 19

2.5.2 Modèle FIFO 19
2.5.3 Mécanismes de QoS 20
2.5.4 Métriques et critères de qualité 21
2.5.5 Garanties de qualité de service 21
2.6 Conclusion 21

3 Le Temps **22**
3.1 Introduction 22
3.2 Temps horloge 22
3.2.1 Définition et fonctionnement des horloges logiques 22
3.3 Temps environnement 23
3.3.1 Impact de l'environnement sur la gestion du temps dans les systèmes répartis 23
3.4 Environnement synchrone 24
3.4.1 Caractéristiques et applications des environnements synchrones . 24
3.5 Temps physique 24
3.5.1 Différences entre temps physique et temps logique dans les systèmes répartis 24
3.6 Horloges physiques synchrones 25
3.6.1 Technologies et protocoles utilisés pour synchroniser les horloges physiques 26
3.7 Conclusion 26

4 Les algorithmes de Concurrence **27**
4.1 Introduction 27
4.1.1 Importance de la Concurrence et de la Cohérence 27
4.2 Algorithmes de Concurrence de Lamport 27
4.2.1 Fondements et Principes de l'Algorithme de Lamport 27
4.2.2 Mécanismes de Contrôle de la Concurrence 28
4.3 Algorithmes de Concurrence de Ricart et Agrawala 28
4.3.1 Compréhension de l'Algorithme de Ricart et Agrawala 28
4.3.2 Comparaison avec d'Autres Approches 29
4.4 Algorithmes de Concurrence de Carvalho et Roucairol 30
4.4.1 Architecture et Fonctionnement de l'Algorithme de Carvalho et Roucairol 31
4.4.2 Applications et Limitations 31
4.5 Études de Cas et Implémentations Réelles 32
4.5.1 Exemples d'Applications Pratiques 32
4.5.2 Déploiement et Intégration dans des Environnements Réels 32
4.6 Comparaison et Analyse des Performances 32
4.6.1 Critères d'Évaluation 33
4.6.2 Scénarios d'Utilisation et Résultats 33
4.7 Analyse de la complexité des algorithmes de concurrence dans les systèmes répartis 33
4.8 Conclusion et Synthese 34
4.8.1 Récapitulatif des Principaux Points 34
4.8.2 Implications et Importance des Algorithmes de Concurrence . . . 34

5 L'observation **38**
5.1 Introduction . . . 38
5.1.1 Définitions et Concepts Clés . . . 38
5.1.2 Importance de l'Observation dans les Systèmes Répartis . . . 38
5.2 État d'une Application Répartie . . . 38
5.2.1 Compréhension de l'État d'une Application Répartie . . . 39
5.2.2 Méthodes et Outils d'Observation de l'État . . . 39
5.3 Détection des Propriétés Stables et Paisibles . . . 39
5.3.1 Définitions et Caractéristiques des Propriétés Stables et Paisibles 40
5.3.2 Techniques de Détection des Propriétés . . . 40
5.4 Études de Cas et Applications Pratiques . . . 41
5.4.1 Exemples d'Applications Réparties Réelles . . . 41
5.4.2 Résultats et Impacts de l'Observation dans les Applications Réparties 42
5.5 Défis et Perspectives Futures . . . 43
5.5.1 Défis Actuels dans l'Observation des Systèmes Répartis . . . 43
5.5.2 Nouvelles Directions de Recherche et Innovations . . . 43
5.6 Conclusion et Récapitulatif des Principaux Points . . . 43

6 Les algorithmes d'élection **44**
6.1 Introduction . . . 44
6.2 Algorithmes d'élection dans les systèmes distribués . . . 44
6.2.1 Algorithme de Chang et Roberts . . . 44
6.2.2 Étapes de l'algorithme . . . 45
6.2.3 Schéma de l'algorithme . . . 45
6.2.4 Pseudo-code de l'algorithme . . . 45
6.2.5 Algorithme de Franklin . . . 46
6.2.6 Étapes de l'algorithme . . . 46
6.2.7 Schéma de l'algorithme . . . 47
6.3 Comparaison des Algorithmes d'Élection . . . 47
6.3.1 Critères de Comparaison . . . 47
6.3.2 Recommandations d'Utilisation . . . 48
6.3.3 Critères d'évaluation . . . 48
6.4 Applications et cas d'utilisation . . . 49
6.4.1 Domaines industriels et scientifiques . . . 49
6.5 Conclusion et perspectives . . . 49
6.5.1 Avancées futures dans le domaine des systèmes distribués . . . 49

7 La mémoire virtuelle répartie et linéarisabilité **50**
7.1 Introduction . . . 50
7.1.1 Définitions de base . . . 50
7.2 Mémoire Virtuelle Répartie . . . 50
7.2.1 Concepts fondamentaux . . . 50
7.2.2 Architectures et modèles . . . 51
7.3 Linéarisabilité dans les Systèmes Répartis . . . 51
7.3.1 Fondements théoriques . . . 51
7.3.2 Linéarisabilité par Exclusion Mutuelle . . . 51
7.3.3 Linéarisabilité avec Gestionnaire Statique . . . 52
7.3.4 Linéarisabilité avec Gestionnaire Dynamique . . . 52

7.4 Propagation des Écritures 53
7.4.1 Mécanismes de Propagation 53
7.4.2 Consistance des Données 54
7.5 Conclusion et Perspectives 54

8 L'autostabilisation 57
8.1 Introduction à l'autostabilisation 57
8.1.1 Concepts de base 57
8.1.2 Historique et évolution 57
8.2 Fondements théoriques de l'autostabilisation 57
8.2.1 Modèles de systèmes répartis 58
8.2.2 Automates finis 58
8.3 Routage auto-stabilisant 58
8.3.1 Principes de base 58
8.4 Algorithmes de Routage Auto-Stabilisants 58
8.5 Gestion de la mémoire virtuelle répartie 60
8.5.1 Nécessité et enjeux 61
8.5.2 Techniques et protocoles 61
8.6 Exclusion mutuelle 61
8.6.1 Problématique de l'exclusion mutuelle 61
8.7 Algorithmes d'Exclusion Mutuelle Auto-Stabilisants 61
8.8 Applications de l'autostabilisation 63
8.8.1 Réseaux de capteurs 63
8.8.2 Systèmes de communication 64
8.9 Défis et perspectives de recherche 64
8.9.1 Limitations actuelles 64
8.9.2 Axes de recherche futurs 64

9 Conclusion Générale 65

10 Exercices sur la Communication 67

11 Exercices sur le Temps 73

12 Exercices sur les Algorithmes de Concurrence 80

13 Exercices sur l'Observation 87

14 Exercices sur les Algorithmes d'Élection 90

15 Exercices sur La mémoire virtuelle répartie et linéarisabilité 92

Table des figures

1.1 Schéma des Modèles architecturaux dans les systèmes répartis 6
1.2 Schéma des Modèles de communication dans les systèmes répartis 6
1.3 Schéma du déploiement Cloud Computing avec IaaS, PaaS, et SaaS . . . 7
1.4 Schéma du déploiement de l'Internet des Objets (IoT) 8

2.1 Schéma du Protocole de Communication 15
2.2 Stratégies de contrôle de flux 17

3.1 Schéma des Horloges Logiques dans les Systèmes Répartis 23
3.2 Schéma de la synchronisation des horloges physiques dans un système réparti 25

4.1 Schéma de l'algorithme de Lamport. 35
4.2 Schéma de l'algorithme de Ricart et Agrawala. 36
4.3 Schéma de l'algorithme de Carvalho et Roucairol avec horloges logiques espacées. 37

5.1 Schéma de l'état d'une application répartie avec collecte et supervision des données 39
5.2 Schéma de Détection des Propriétés Stables et Paisibles d'une Application Répartie 41

6.1 Schéma de l'algorithme de Chang et Roberts 45
6.2 Schéma de l'algorithme de Franklin 47

7.1 Mécanismes de Propagation dans les Systèmes Distribués 55
7.2 Consistance des Données dans les Systèmes Distribués 56

12.1 Schéma explicatif du modèle Producteur-Consommateur 81
12.2 Schéma explicatif de la synchronisation dans les systèmes concurrents . . 82

15.1 Schéma d'un Protocole de Consistance dans un Système de Mémoire Virtuelle Répartie (Amélioré) 93
15.2 Schéma de la Gestion de la Mémoire Virtuelle dans un Système Réparti (Amélioré) 94

Introduction Générale

L'évolution rapide des technologies de l'information et de la communication au cours des deux dernières décennies a profondément modifié de manière significative et irréversible notre manière d'interagir avec notre environnement à la fois personnel et professionnel. Au centre de cette transformation majeure se retrouvent les systèmes répartis, qui facilitent non seulement l'interconnexion mais également la collaboration de divers dispositifs, serveurs, et infrastructures au sein de réseaux de plus en plus complexes. Dans un contexte technologique où il est impératif de traiter des volumes massifs de données, tout en fournissant des services en temps réel, une compréhension approfondie des algorithmes et des principes régissant ces systèmes apparaît comme étant d'une importance capitale et incontournable. Le présent document a pour objectif ambitieux de servir de référence exhaustive aux étudiants de Master 1 en Réseaux et Systèmes d'Information Distribués (RSID), en Systèmes d'Information Décisionnels (SID), ainsi qu'en Intelligence Artificielle et Analyse de Données (IAA). Il vise à les familiariser et à les initier avec les concepts fondamentaux des Algorithmes et Systèmes Répartis (ASR) qui sont essentiels dans leur formation.

Les systèmes répartis se définissent clairement comme étant un ensemble d'entités autonomes et distinctes qui échangent des informations mouvantes dans le but d'atteindre des objectifs communs bien définis. Cette configuration offre une flexibilité impressionnante et une résilience sans pare, permettant aux organisations de tirer pleinement parti de ressources distribuées, d'optimiser leurs opérations quotidiennes, mais aussi d'assurer une meilleure continuité de service en cas de défaillance ou de problème imprévu. Ce document explorera en profondeur les divers modèles de systèmes répartis, notamment les architectures client-serveur, les systèmes pair-à-pair, ainsi que les systèmes fondés sur le cloud. Chacune de ces architectures présente des avantages notables mais également des inconvénients qu'il sera crucial que les étudiants maîtrisent. Cela leur permettra de prendre des décisions éclairées lors de la conception de solutions adaptées à des problématiques variées et spécifiques rencontrées dans leur futur professionnel. L'une des dimensions les plus fascinantes et stimulantes des systèmes répartis réside incontestablement dans la diversité des algorithmes susceptibles d'être appliqués pour résoudre des problèmes spécifiques et complexes. Au cours de ce cursus enrichissant, les étudiants seront initiés à des algorithmes fondamentaux tels que ceux de consensus, de répartition de charge, de synchronisation, et de détectio de pannes. Chacun de ces algorithmes est essentiel pour garantir la cohérence, la fiabilité et l'efficacité des systèmes répartis, facilitant ainsi leur fonctionnement quotidien. En étudiant ces algorithmes, les étudiants apprendront non seulement à les mettre en œuvre de façon pratique, mais aussi à les évaluer rigoureusement selon des critères de performance et de complexité technique.

Parallèlement à l'étude des algorithmes, ce module abordera également les défis associés à la gestion et l'administration des systèmes répartis, tels que la tolérance aux pannes, la sécurité des données, ainsi que la gestion des transactions en cours. Dans un

environnement actuel où les menaces cybernétiques sont omniprésentes et de plus en plus sophistiquées, il est crucial que les futurs professionnels possèdent des compétences approfondies en matière de sécurité des systèmes. Les discussions détaillées concernant la sécurité porteront sur les meilleures pratiques et stratégies pour protéger les données sensibles et garantir l'intégrité des systèmes tout en maintenant un niveau élevé de disponibilité opérationnelle. L'intégration des concepts d'intelligence artificielle et d'apprentissage automatique dans le domaine des systèmes répartis représente une tendance émergente et captivante, qui suscite un intérêt grandissant au sein du milieu académique et professionnel. Au cours de ce module, nous allons analyser en détail comment les techniques d'intelligence artificielle peuvent être appliquées pour optimiser les systèmes distribués. Cela inclut notamment l'optimisation des processus décisionnels ainsi que la prévision des défaillances, qui constitue un axe fondamental d'étude pour les étudiants. Ces derniers auront également l'opportunité précieuse d'analyser des cas d'utilisation concrets où l'intelligence artificielle et les systèmes répartis se combinent habilement afin de générer des solutions à la fois innovantes et réellement efficaces.

Ce polycopié sera aussi complété par des études de cas pratiques, des travaux dirigés, et des projets collaboratifs qui permettront aux étudiants de mettre en application les connaissances acquises tout au long de la formation. L'apprentissage par la pratique reste crucial pour développer une compréhension approfondie des concepts théoriques et pour se préparer efficacement à des défis concrets. Les étudiants seront activement encouragés à collaborer en groupe, à partager leurs idées, à débattre et à élaborer des solutions collectives qui illustrent les dynamiques des systèmes répartis modernes. En résumé, le module "Algorithmes et Systèmes Répartis" a pour objectif de fournir aux étudiants une formation rigoureuse, exhaustive et richement détaillée dans un domaine clé du paysage technologique contemporain en constante évolution. En acquérant des compétences techniques et analytiques avancées, les étudiants seront ainsi mieux armés pour faire face aux défis futurs rencontrés dans divers secteurs tels que les télécommunications, l'industrie 4.0, l'Internet des objets, ainsi que dans bien d'autres domaines encore. Ce polycopié représente un outil essentiel et précieux dans leur parcours académique et professionnel, les aidant à devenir des acteurs majeurs et influents dans le développement de solutions innovantes et performantes au sein des systèmes répartis.

Chapitre 1

Généralités sur les systèmes répartis

Les systèmes distribués peuvent être décrits comme des ensembles de composants indépendants et interconnectés, lesquels communiquent et collaborent dans le but d'atteindre des objectifs partagés. Contrairement aux systèmes centralisés, où le contrôle et les ressources sont concentrés en un unique point, les systèmes distribués se distinguent par leur répartition tant géographique que logique. Cette architecture favorise une meilleure tolérance aux pannes, une évolutivité supérieure et une flexibilité accrue dans le déploiement des ressources. Les avancées technologiques ainsi que l'essor d'Internet ont largement contribué à la popularisation de ces systèmes, permettant le développement d'applications variées, allant des réseaux sociaux aux plateformes de cloud computing.

Au centre des systèmes répartis se trouvent plusieurs concepts fondamentaux, parmi lesquels la transparence, la concurrence et la gestion des ressources. La transparence a pour objectif de dissimuler la complexité ainsi que les détails relatifs à la répartition des ressources vis-à-vis des utilisateurs et des applications, facilitant ainsi une interaction plus intuitive. La concurrence, pour sa part, aborde l'accès simultané aux ressources partagées, un élément crucial pour la performance et l'efficacité des applications. La gestion des ressources comprend l'allocation, la planification et la supervision de divers composants du système, veillant à établir un équilibre entre la performance et une utilisation optimale des ressources. Ces défis techniques ont engendré le développement d'algorithmes spécifiques, tels que ceux employés dans la synchronisation des processus et la gestion des transactions, garantissant qu'en dépit de la nature décentralisée des systèmes, les opérations demeurent cohérentes et fiables.

L'importance des systèmes répartis va au-delà de leur architecture technique. Ils induisent également une transformation profonde dans notre manière de concevoir et de déployer des services. À mesure que les entreprises adoptent des infrastructures décentralisées, elles se voient dans l'obligation d'examiner des enjeux relatifs à la sécurité, à la gestion de la performance, ainsi qu'à l'interopérabilité. Par ailleurs, l'implémentation de ces systèmes soulève des interrogations éthiques et sociétales, notamment en ce qui concerne la protection des données personnelles et l'accès équitable aux ressources numériques. En résumé, les systèmes répartis incarnent une rencontre captivante de technologies, de pratiques et de considérations sociales, mettant en exergue la nécessité d'une approche multidisciplinaire pour leur gestion et leur développement.

1.1 Introduction aux systèmes répartis

Les systèmes distribués représentent un domaine fondamental de l'informatique, englobant des ensembles de composants autonomes et interconnectés, qui collaborent pour accomplir des tâches de manière coopérative. Leur conception s'appuie sur des principes d'architecture distribuée, où chaque nœud peut être situé sur un même site ou dispersé sur différents emplacements géographiques, permettant ainsi une augmentation notable des capacités de traitement, de stockage et de fiabilité des systèmes informatiques. L'interopérabilité constitue une caractéristique essentielle des systèmes distribués, permettant à des unités variées de fonctionner ensemble grâce à des protocoles de communication uniformisés qui régulent l'échange d'informations et les interactions entre les composants.

Les applications des systèmes distribués sont diverses et se manifestent dans de nombreux domaines, allant des systèmes de fichiers distribués aux architectures de services web, ainsi que le cloud computing. Ce dernier a radicalement modifié la manière dont les ressources informatiques sont allouées et utilisées, en proposant des infrastructures flexibles et scalables à la demande. Par ailleurs, les enjeux liés à la gestion des pannes, de la latence et de la cohérence des données nécessitent des approches novatrices pour le développement d'algorithmes appropriés. Ces algorithmes comprennent des techniques telles que le consensus distribué et la réplication, qui sont essentielles pour garantir la continuité des opérations des systèmes, même en cas de défaillances de nœuds ou de réseaux. La sécurité et la sauvegarde des données dans un environnement distribué constituent des enjeux essentiels. Il est impératif que les systèmes soient élaborés de manière à pouvoir faire face à des menaces potentielles, ce qui implique l'instauration de mécanismes de sécurité solides, tels que le chiffrement des échanges et la gestion des accès. Par conséquent, l'importance d'une conception adéquate des systèmes distribués ne se limite pas uniquement à leur efficacité et à leur fiabilité, mais s'étend également à la préservation des informations traitées. En définitive, les systèmes distribués offrent des solutions puissantes et évolutives, qui continuent de s'adapter aux exigences croissantes en matière de traitement des données et de services dans un environnement interconnecté.

1.1.1 Définition et concepts clés

La définition des systèmes répartis repose sur le principe fondamental des ressources partagées, ainsi que sur la communication efficace entre des entités distinctes qui sont interconnectées. Les concepts fondamentaux qui sous-tendent ces systèmes englobent la transparence, la concurrence, l'évolutivité et la tolérance aux pannes, chacun jouant un rôle crucial dans le fonctionnement des systèmes. La transparence, par exemple, se réfère à la manière dont un système réparti parvient à dissimuler la complexité de sa structure interne et de son fonctionnement détaillé auprès des utilisateurs finaux. Assimiler ces définitions ainsi que ces concepts clés est essentiel pour permettre une exploration approfondie et riche des systèmes répartis, tout en comprenant les enjeux et les défis qui peuvent en découler.

1.1.2 Avantages et inconvénients

Les systèmes répartis, avec leur architecture décentralisée, offrent des avantages et des inconvénients significatifs. Parmi les bénéfices, la scalabilité et la tolérance aux pannes sont essentielles, permettant l'ajout de nouveaux nœuds facilement et maintenant les

opérations malgré les pannes. Toutefois, ces atouts s'accompagnent de challenges notables. La complexité de gestion est une préoccupation majeure, car la synchronisation entre nœuds complique leur conception et maintenance. Les délais de communication réseau peuvent également nuire à la réactivité. Les mesures de sécurité sont renforcées par l'accès décentralisé, entraînant des coûts supplémentaires. De plus, des incohérences de données peuvent apparaître, rendant nécessaires des mécanismes pour assurer leur cohérence. Bien que flexibles et résilients, ces systèmes exigent une analyse rigoureuse des défis techniques à relever et des choix d'implémentation basés sur une compréhension claire des avantages et désavantages pour optimiser l'efficacité tout en minimisant les risques.

1.2 Modèles de répartition

Les modèles de répartition sont essentiels dans les systèmes distribués, gérant ressources et tâches. Ils influencent l'allocation des ressources et les interactions des composants. Les principaux enjeux sont la scalabilité, la tolérance aux pannes et l'efficacité des communications. Les approches varient des modèles centralisés à des modèles décentralisés avec des nœuds autonomes. Certains utilisent la géolocalisation pour réduire les latences, plaçant les ressources selon l'emplacement des utilisateurs, tandis que d'autres optimisent la charge de travail en redistribuant les tâches. Les modèles hybrides gagnent en popularité pour leur flexibilité. La validation de ces modèles est cruciale, souvent par simulations et tests réels. Les avancées en intelligence artificielle offrent de nouvelles opportunités pour optimiser ces modèles, permettant des décisions rapides. En résumé, les modèles de répartition sont fondamentaux pour le partage des ressources, structurant la conception des systèmes distribués et évaluant leurs performances.

1.2.1 Modèles architecturaux

Les modèles architecturaux sont essentiels dans le développement des systèmes répartis, organisant les composants et leurs interactions. Un modèle architectural est une abstraction simplifiant l'analyse de performance et de fiabilité. Divers modèles existent pour des exigences spécifiques. Le modèle client-serveur, bien que performant, limite la scalabilité avec l'augmentation des clients. En revanche, l'architecture P2P, où chaque nœud agit en tant que client et serveur, améliore la résilience et la tolérance aux pannes, mais requiert une gestion stricte de la sécurité. Les microservices découpent les applications en services autonomes, offrant agilité tout en nécessitant une infrastructure adéquate, comme des conteneurs. Une conception architecturale solide est cruciale pour l'efficacité et la robustesse des systèmes distribués, facilitant leur évolution. Ces modèles définissent la structure du système, la répartition des composants et leurs connexions, et clarifient les types de nœuds et leurs fonctions.

1.2.2 Modèles de communication

Les modèles de communication dans les systèmes répartis sont essentiels pour optimiser l'échange d'informations. Les modèles par messages assurent une communication asynchrone, tandis que les flux de données garantissent rapidité et synchronisation. L'abstraction est cruciale, car des modèles simples comme le client-serveur peuvent créer

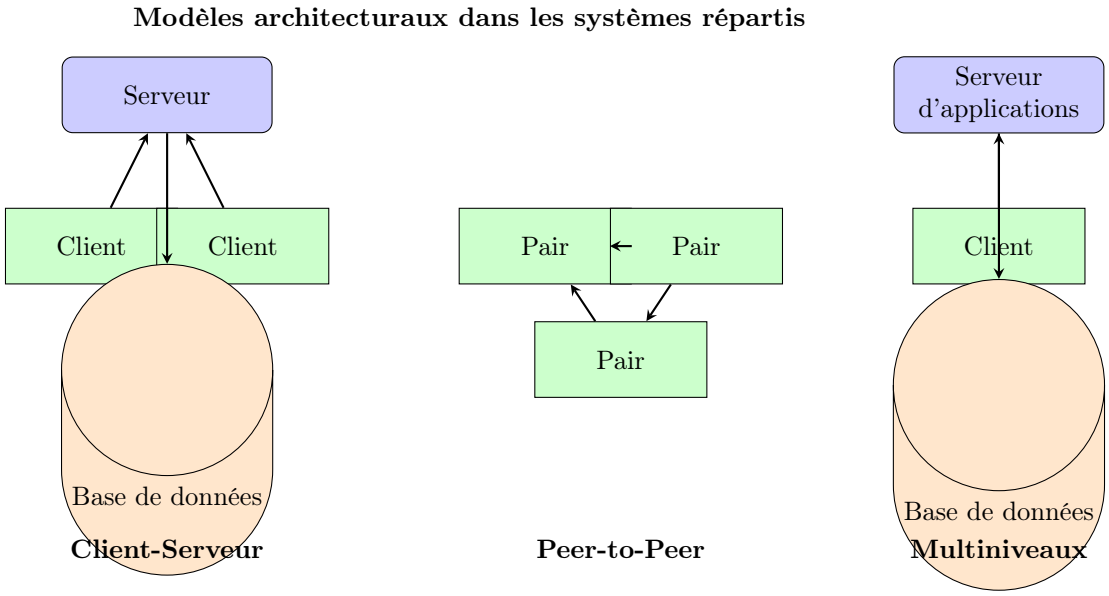


FIGURE 1.1 – Schéma des Modèles architecturaux dans les systèmes répartis

des goulots d'étranglement, limitant la scalabilité. Les systèmes réactifs et l'architecture orientée services (SOA) facilitent la communication dynamique via des protocoles standardisés, améliorant la répartition des tâches. Les microservices, opérant de manière autonome par le biais d'API, montrent l'importance d'adopter ces modèles pour bâtir des systèmes solides. Le choix du modèle impacte traitement des données et résilience. Les systèmes futurs, intégrant l'intelligence artificielle, perfectionneront les interactions dans des réseaux complexes. Ces modèles répondent à divers besoins tout en tenant compte de la synchronisation, de la gestion des erreurs et de la sécurité.

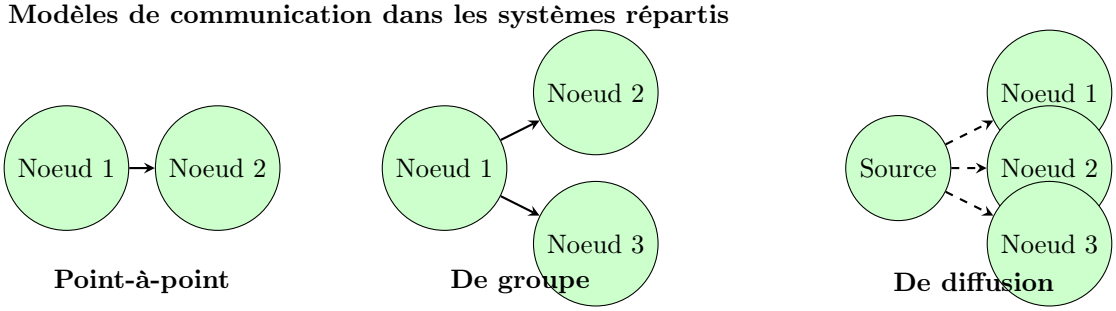


FIGURE 1.2 – Schéma des Modèles de communication dans les systèmes répartis

1.3 Scénarios de déploiement

Les scénarios de déploiement des systèmes répartis optimisent les architectures selon la localisation et les capacités des nœuds. Le cloud public convient aux applications à forte demande, tandis que le déploiement sur site est préférable pour la sécurité des données, notamment dans le secteur bancaire. Le choix dépend du type d'application : les systèmes en temps réel nécessitent une faible latence avec des configurations comme des clusters géographiques proches. Les microservices améliorent la résilience et la gestion des

défaillances. Les scénarios hybrides, combinant cloud public et privé, sont en hausse en raison de la conformité. Les outils de gestion offrent visibilité et performance. Il est essentiel d'aligner les déploiements avec les objectifs stratégiques pour garantir performance et fiabilité.

1.3.1 Cloud computing

Le cloud computing a révolutionné l'accès aux ressources informatiques via Internet, remplaçant l'infrastructure locale. Il comprend l'IaaS pour la location de ressources virtualisées, le PaaS pour le développement d'applications et le SaaS pour des applications en ligne. La scalabilité ajuste les ressources selon la demande, et la sécurité est cruciale pour les fournisseurs. Économique, le cloud permet un paiement à l'usage, réduisant les coûts. Les systèmes distribués renforcent la tolérance aux pannes, tandis que l'informatique en périphérie et les conteneurs réduisent la latence. L'IA et l'analyse des big data optimisent le processus décisionnel et le développement des services numériques.

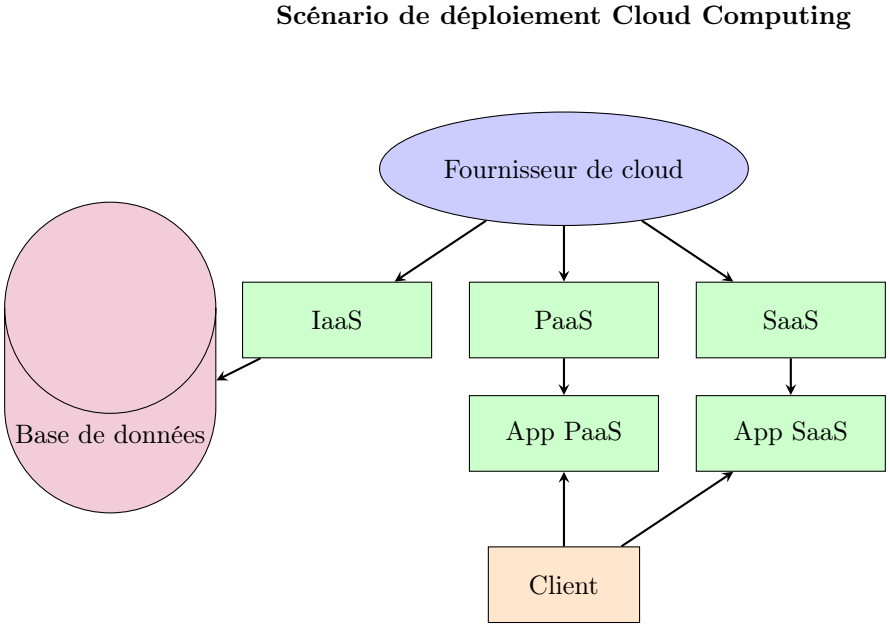


FIGURE 1.3 – Schéma du déploiement Cloud Computing avec IaaS, PaaS, et SaaS

1.3.2 Internet des objets (IoT)

L'Internet des objets (IoT) connecte des objets physiques pour collecter des données via des capteurs et logiciels, créant un écosystème d'informations en temps réel. Cela est utile dans des secteurs comme la gestion de l'énergie, la domotique, et la santé connectée. Des protocoles comme MQTT et CoAP permettent le transfert de données, tandis que la sécurité est cruciale pour protéger les informations. Les réseaux de capteurs sans fil (WSN) facilitent la communication à distance et les technologies cloud et edge computing sont essentielles pour le traitement des données. L'IoT révolutionne les entreprises, particulièrement en santé et agriculture, en optimisant les processus et en assurant la connectivité sécurisée pour des milliards d'appareils interconnectés.

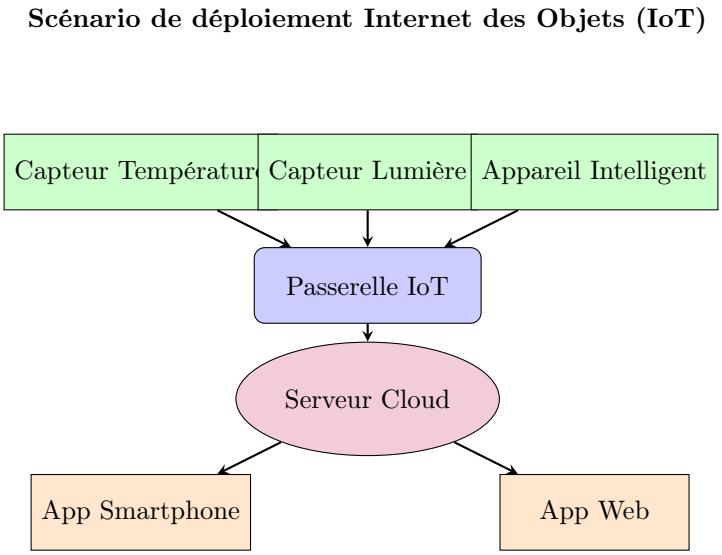


FIGURE 1.4 – Schéma du déploiement de l'Internet des Objets (IoT)

1.4 Évaluation des performances

L'évaluation des performances au sein des systèmes répartis revêt une importance capitale pour appréhender leur comportement dans des contextes variés. Ceci englobe l'analyse des temps de réponse, de la capacité de traitement, de la disponibilité du système et de sa fiabilité. Ces paramètres facilitent l'identification des secteurs nécessitant des améliorations et assurent le bon fonctionnement optimal du système.

1.4.1 Critères d'évaluation

Les critères d'évaluation des systèmes répartis incluent la précision, la fiabilité, la scalabilité, la disponibilité et l'efficacité. La précision concerne la justesse, la fiabilité la stabilité, tandis que la scalabilité évalue l'adaptation à l'augmentation des utilisateurs. La disponibilité se concentre sur le temps d'opération, et l'efficacité sur les ressources. Les algorithmes doivent être rapides et adaptables, tout en maintenant la cohérence des données, même lors des interruptions. La sécurité est essentielle pour la protection des données.

1.4.2 Outils de mesure

Les instruments de mesure des performances des systèmes distribués comprennent profilers, dispositifs de surveillance, simulateurs de charge et analyseurs de trafic. Les profilers identifient le code gourmand, les surveillants examinent les flux de données, et les simulateurs testent la réactivité. Des outils comme Prometheus, Grafana et Apache JMeter sont utilisés pour recueillir des métriques, les visualiser et tester la charge. Ces outils permettent aux ingénieurs de diagnostiquer efficacement les problèmes.

Algorithm 1 Évaluation des performances

Entrée : Liste des nœuds $N = \{n_1, n_2, \ldots, n_k\}$, liste des connexions $C = \{c_1, c_2, \ldots, c_m\}$
Sortie : Performance globale du système P
$P \leftarrow 0$ ▷ Initialisation de la performance
Initialisation des critères :
    Latence $L \leftarrow 0$
    Bande passante $B \leftarrow 0$
    Fiabilité $R \leftarrow 0$
    Temps de disponibilité $U \leftarrow 0$
    Coût des ressources $Cost \leftarrow 0$
for chaque connexion $c \in C$ **do**
    $L \leftarrow L +$ Mesurer la latence de c
    $B \leftarrow B +$ Mesurer la bande passante de c
    $R \leftarrow R +$ Calculer la fiabilité de c
end for
for chaque nœud $n \in N$ **do**
    $U \leftarrow U +$ Mesurer le temps de disponibilité de n
    $Cost \leftarrow Cost +$ Évaluer le coût de n
end for
Calcul de la performance globale :
$P \leftarrow \alpha_1 \cdot \frac{L}{m} + \alpha_2 \cdot \frac{B}{m} + \alpha_3 \cdot \frac{R}{m} + \alpha_4 \cdot \frac{U}{k} - \alpha_5 \cdot \frac{Cost}{k}$
▷ Interprétation de P :
    Si $P > T$ (*seuil fixé*) **alors** la performance est jugée **bonne**.
    Sinon, la performance est jugée **insatisfaisante**.
return P

1.5 Vérification des algorithmes répartis

La vérification des algorithmes répartis est cruciale pour la fiabilité des systèmes décentralisés, nécessitant des méthodes formelles pour détecter les anomalies. L'analyse des interactions asynchrones complique cette validation, nécessitant gestion de la concurrence et synchronisation des nœuds. Il est indispensable d'évaluer correction et tolérance aux pannes. La vérification formelle et la simulation d'algorithmes sont essentielles, même si les tests ne garantissent pas des résultats complets. Des outils doivent être intégrés pour détecter les erreurs avant le déploiement.

1.5.1 Méthodes formelles

Les approches formelles pour vérifier les algorithmes répartis s'appuient sur des principes mathématiques pour prouver leur correction. Elles englobent la modélisation formelle, les démonstrations de programmes et la logique de Hoare, validant ainsi la fiabilité des systèmes distribués. Les logiques temporelles aident à établir des spécifications pour l'analyse et valider des propriétés comme la terminaison. Les outils de vérification explorent l'espace d'état, identifiant les comportements indésirables, renforçant la confiance dans le développement logiciel, essentiel dans des domaines critiques comme l'aéronautique

1.5.2 Vérification de la cohérence

La vérification de la cohérence des algorithmes répartis est un aspect crucial de l'analyse formelle. Elle vise à s'assurer que les différentes parties du système fonctionnent de manière cohérente et en harmonie les unes avec les autres. Cela implique d'identifier et de résoudre les éventuels conflits et incohérences dans les calculs et les décisions prises par les composants distribués. Les techniques de vérification de la cohérence reposent sur des modèles spécifiques et des analyses approfondies pour garantir l'intégrité et la cohérence globale du système réparti.

1.6 Conclusion

Dans ce premier chapitre, nous avons exploré les bases des systèmes répartis, un domaine clé pour comprendre la manière dont les ressources et les tâches peuvent être réparties entre plusieurs entités interconnectées. Nous avons défini les concepts fondamentaux tels que la transparence, la concurrence, la scalabilité et la tolérance aux pannes, qui constituent les piliers de la conception et de la gestion de ces systèmes.

L'étude des avantages et des inconvénients des systèmes répartis nous a permis de mettre en lumière leur potentiel en termes de fiabilité, flexibilité et efficacité des ressources, tout en soulignant les défis complexes liés à leur sécurité et leur gestion. De plus, l'examen des différents modèles de répartition architecturale et des communications a mis en évidence les structures et mécanismes sous-jacents qui facilitent la coopération entre les différents composants du système.

Cette introduction nous prépare aux chapitres suivants où nous approfondirons les aspects plus techniques des systèmes répartis, notamment les algorithmes de coordination, la gestion des pannes et les scénarios avancés de déploiement tels que l'Internet des objets

et le cloud computing. Le but est d'acquérir une compréhension complète de ces systèmes afin de les concevoir et de les utiliser de manière efficace et sécurisée.

Chapitre 2

La communication

2.1 Introduction

Les systèmes répartis font référence à des collections de ressources informatiques qui agissent comme un seul système cohérent. Ce type de système est capable de coordonner l'utilisation des ressources distribuées pour répondre à un ensemble d'objectifs communs. Il est essentiel de comprendre les défis uniques liés à la conception et à la gestion de tels systèmes. Les systèmes répartis peuvent inclure des applications telles que les réseaux de capteurs sans fil, les systèmes de traitement distribué et les bases de données réparties.

2.1.1 Applications des systèmes répartis

Les systèmes répartis sont largement utilisés dans les environnements informatiques modernes. Ils sont utilisés pour la mise en réseau d'entreprises, les services cloud, les applications mobiles et Internet des objets. Ces systèmes offrent une flexibilité et une évolutivité supérieures, permettant aux organisations de répondre plus efficacement aux besoins changeants de leurs utilisateurs et de leurs clients.

2.2 Fondements de la communication

Les fondements de la communication dans les systèmes répartis reposent sur la nécessité de transmettre des données entre les différents composants du système de manière efficace et fiable. Cela inclut la prise en compte de la distance entre les nœuds, la tolérance aux pannes et la gestion des ressources. La modélisation de la communication dans les systèmes répartis implique l'utilisation de modèles tels que le modèle client-serveur, le modèle Peer-to-Peer ou le modèle de bus. Ces modèles permettent de décrire les interactions entre les entités du système et de définir les rôles et les responsabilités de chacune. Les protocoles de communication, quant à eux, spécifient les règles et les formats à suivre pour assurer une transmission adéquate des données. Enfin, les mécanismes de communication comprennent les techniques de partage de ressources, de synchronisation et de gestion de la latence, qui sont essentielles pour assurer des échanges de données performants dans un environnement réparti.

2.2.1 Modèles de communication

Les modèles de communication utilisés dans les systèmes répartis sont des structures conceptuelles qui définissent les interactions entre les différents composants du système. Le modèle client-serveur est largement utilisé dans les environnements où un client envoie des requêtes à un serveur pour obtenir des services. Le modèle Peer-to-Peer permet aux nœuds de communiquer directement les uns avec les autres sans passer par un serveur central. Le modèle de bus consiste à connecter tous les composants du système à un bus de communication partagé. Chaque modèle présente des avantages et des inconvénients en termes de performances, de flexibilité et de tolérance aux pannes, ce qui les rend adaptés à des cas d'utilisation spécifiques.

2.2.2 Protocoles de communication

Les protocoles de communication définissent les règles et les formats à suivre pour une transmission de données efficace dans les systèmes répartis. Cela inclut la manière dont les données sont structurées, encodées, envoyées, reçues et interprétées. Les principaux protocoles de communication utilisés dans les systèmes répartis incluent TCP/IP pour la communication sur Internet, HTTP pour la communication web, et MQTT pour la communication machine à machine. Chaque protocole est conçu pour répondre à des besoins spécifiques en matière de fiabilité, de vitesse, de sécurité et de gestion des erreurs.

2.2.3 Mécanismes de communication

Les mécanismes de communication dans les systèmes répartis comprennent un large éventail de techniques visant à assurer des échanges de données efficaces. Cela inclut la gestion des ressources telles que la bande passante et le stockage, la synchronisation des opérations entre les différents nœuds, la gestion des files d'attente pour réguler le flux de données, et la garantie de l'ordre d'arrivée des messages. Les mécanismes de communication sont essentiels pour garantir la cohérence, la fiabilité et les performances des échanges de données dans un environnement distribué.

2.3 Contrôle de flux dans les systèmes répartis

Le contrôle de flux dans les systèmes répartis fait référence à la gestion du flux de données entre les différents composants d'un système distribué. Cela implique de surveiller et de réguler la vitesse à laquelle les données sont envoyées et reçues afin d'éviter la congestion du réseau et les pertes de données. Les stratégies de contrôle de flux peuvent être basées sur des approches telles que le contrôle de fenêtre, le contrôle de taux, ou le contrôle de crédit. En implémentation, des algorithmes tels que le contrôle de débit, les tampons de réception et les fenêtres d'envoi peuvent être utilisés pour gérer efficacement le flux de données. Des exemples concrets de contrôle de flux incluent le protocole TCP qui utilise un mécanisme de contrôle de congestion pour maintenir un flux de données stable dans un réseau distribué.

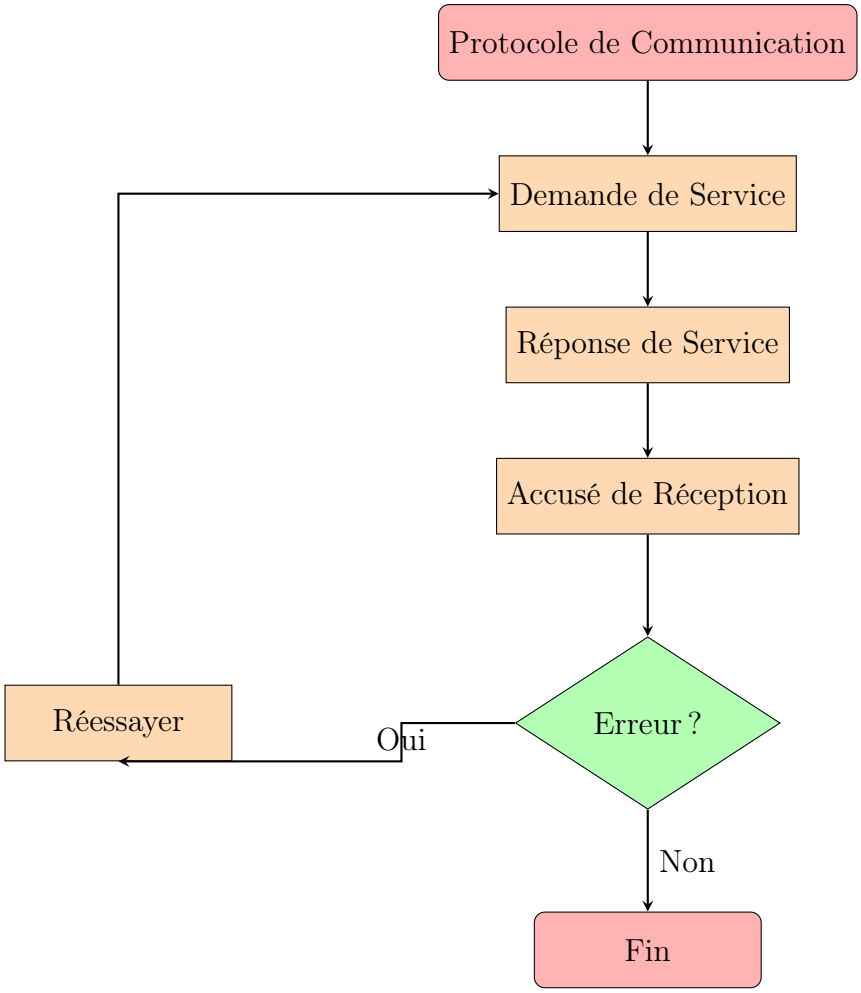


FIGURE 2.1 – Schéma du Protocole de Communication

2.3.1 Notions de base

Les notions de base du contrôle de flux dans les systèmes répartis comprennent la compréhension des concepts tels que le flux de données, la vitesse de transmission, la congestion du réseau et les pertes de données. Il s'agit également de comprendre les mécanismes de communication utilisés pour contrôler le flux de données, tels que le contrôle de fenêtre et le contrôle de taux. De plus, la connaissance des algorithmes et des protocoles de contrôle de flux tels que TCP et UDP est essentielle pour mettre en place des stratégies de gestion efficaces dans un environnement distribué.

2.3.2 Stratégies de contrôle de flux

Les stratégies de contrôle de flux dans les systèmes répartis comprennent des approches telles que le contrôle de fenêtre coulissante, le contrôle de taux d'envoi et le contrôle de crédit. Le contrôle de flux adaptatif peut également être utilisé pour ajuster dynamiquement la vitesse d'envoi en fonction des conditions du réseau. D'autres stratégies incluent l'utilisation de tampons pour gérer les fluctuations de débit et la mise en œuvre de politiques de retransmission pour gérer les pertes de données. Ces stratégies visent à optimiser la performance et la fiabilité du flux de données dans les systèmes répartis.

Algorithm 2 Stratégie de contrôle de flux : Fenêtre Glissante

```
1: N ← taille maximale de la fenêtre
2: S ← 0                                          ▷ Séquence des trames envoyées
3: R ← 0                                          ▷ Séquence des trames reçues
4: A ← 0                                          ▷ Dernier accusé de réception reçu
5: T ← timeout                                    ▷ Valeur du timeout
6: while S < A + N do
7:     Envoyer_trame(S)
8:     Démarrer_timer_pour_trame(S)               ▷ Ajouté selon remarque
9:     S ← S + 1
10: end while
11: while Réception d'une trame do
12:     if Trame valide reçue then
13:         R ← numéro de séquence de la trame reçue
14:         Envoyer_ACK(R)
15:     else
16:         Réémettre_ACK_dernière_trame_valide()
17:     end if
18: end while
19: while Accusé de réception reçu do
20:     A ← numéro ACK reçu
21:     if A == S then
22:         Fin de transmission si toutes les trames ont été reçues ▷ Clarification
    ajoutée
23:     end if
24: end while
25: if Timeout atteint then
26:     Réémettre_trame(A + 1)
27:     Réinitialiser compteur de délai       ▷ Le compteur de délai est le timeout
28: end if
29: T ← timeout                                   ▷ Valeur du timeout
30: Réinitialiser compteur de délai               ▷ Le compteur de délai est le timeout
```

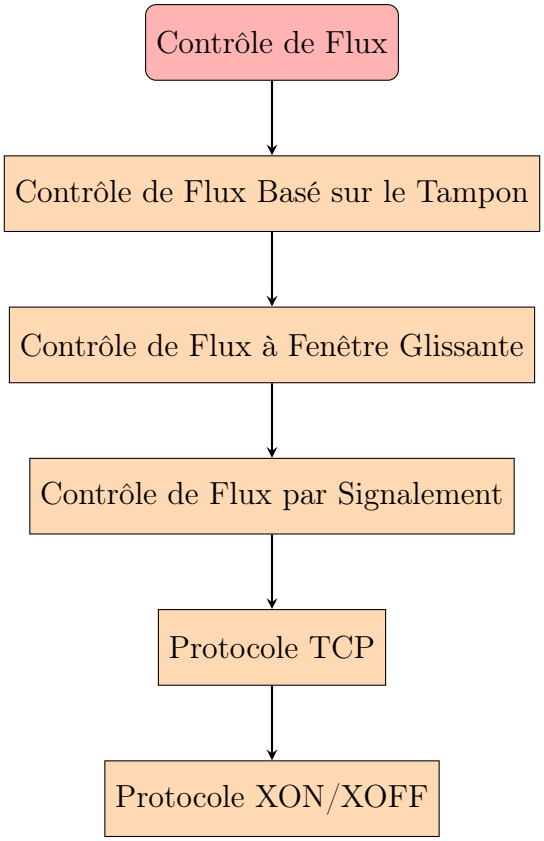


FIGURE 2.2 – Stratégies de contrôle de flux

2.3.3 Implémentation et exemples

L'implémentation du contrôle de flux dans les systèmes répartis peut se faire à l'aide d'algorithmes tels que le contrôle de congestion TCP, qui ajuste la fenêtre de congestion en fonction des signaux du réseau. Des exemples concrets d'implémentation incluent l'utilisation de mécanismes de contrôle de flux dans les applications de diffusion vidéo en continu pour garantir une lecture sans heurt, ainsi que dans les systèmes de stockage distribué pour gérer le flux de données entrant et sortant de manière efficace. Ces exemples mettent en évidence l'importance du contrôle de flux dans la garantie de la performance et de la fiabilité des systèmes répartis.

2.4 Communication synchrone avec RdV

La communication synchrone avec rendez-vous (RdV) est un type de communication dans les systèmes répartis où les processus doivent se coordonner pour échanger des informations. Ce type de communication présente l'avantage d'assurer une synchronisation stricte entre les processus, ce qui peut être crucial dans de nombreux cas d'utilisation. Cependant, cela peut également entraîner des performances plus faibles par rapport à la communication asynchrone, en raison de la nécessité de synchroniser les processus. Il est donc important de bien comprendre les caractéristiques de ce type de communication avant de l'implémenter dans un système distribué.

2.4.1 Caractéristiques de la communication synchrone

La communication synchrone se caractérise par la coordination temporelle des processus impliqués, qui doivent être actifs et prêts à échanger des données au même moment. Cela nécessite un certain degré de synchronisation pour garantir que les processus se rencontrent au bon moment. De plus, la communication synchrone implique souvent l'utilisation de structures de données spécifiques et la définition de protocoles pour gérer le rendez-vous et l'échange de données. Ces caractéristiques font de la communication synchrone un choix approprié pour certains cas d'utilisation nécessitant une coordination stricte entre les processus.

2.4.2 Rendez-vous et synchronisation

Le rendez-vous dans la communication synchrone se réfère à la synchronisation des processus pour qu'ils se rencontrent et échangent des données de manière coordonnée. Cela peut être réalisé à l'aide de mécanismes de synchronisation tels que les sémaphores, les moniteurs ou les verrous. La synchronisation est essentielle pour garantir que les processus ne se rencontrent pas de manière aléatoire, mais au contraire dans un ordre précis et prévisible. L'utilisation de mécanismes de rendez-vous permet également d'éviter les blocages et d'assurer une communication fluide entre les processus.

Algorithm 3 Protocole de communication synchrone avec RdV (Rendez-vous)

1: **Variables :**
2: $M \leftarrow$ message à envoyer
3: $ACK \leftarrow false$ ▷ Accusé de réception
4: $timeout \leftarrow T_{\max}$ ▷ Valeur maximale du timer
5: **Processus A (Expéditeur)**
6: **while** $ACK == false$ **do**
7: Envoyer_message(M) ▷ Envoie du message à B
8: Démarrer_timer($timeout$) ▷ Ajouté pour gérer l'expiration du timer
9: Attendre_ACK() ▷ Attendre l'accusé de réception
10: **if** Réception de ACK **then**
11: $ACK \leftarrow true$ ▷ Confirmation de la réception
12: **else if** Timer_expiré() **then**
13: Réémettre_message(M) ▷ Réémettre en cas de timeout
14: **end if**
15: **end while**
16: **Processus B (Récepteur)**
17: **while** Réception de message **do**
18: Lire_message(M)
19: Envoyer_ACK() ▷ Envoyer l'accusé à A
20: **end while**

2.4.3 Implémentation de la communication synchrone

L'implémentation de la communication synchrone dans les systèmes répartis peut être réalisée à l'aide de bibliothèques et de frameworks de communication qui facilitent la gestion des rendez-vous et des échanges de données. Des exemples d'outils incluent MPI

(Message Passing Interface), qui offre des primitives pour la communication synchrone entre les processus, ainsi que des langages de programmation qui prennent en charge la programmation concurrente et la synchronisation. L'utilisation de ces outils permet de simplifier le développement d'applications distribuées nécessitant une communication synchrone.

2.4.4 Applications et cas d'utilisation

La communication synchrone est souvent utilisée dans des applications critiques où la coordination entre les processus est primordiale. Cela inclut des systèmes de contrôle industriel, des applications financières en temps réel, et des systèmes de communication en réseau. Dans ces cas d'utilisation, la communication synchrone garantit que les données sont échangées de manière fiable et en temps voulu, ce qui est essentiel pour le bon fonctionnement des applications. L'utilisation de la communication synchrone peut également améliorer la fiabilité et la cohérence des systèmes répartis en garantissant une synchronisation appropriée entre les processus.

2.5 Qualité de service : réseau FIFO

La qualité de service (QoS) est un aspect essentiel des systèmes répartis, car elle détermine la capacité du système à fournir des services de manière fiable et efficace. Un réseau de qualité de service est conçu pour assurer des performances optimales en garantissant la bande passante, la latence, et la fiabilité. Le modèle FIFO (First In, First Out) est souvent utilisé pour gérer les files d'attente et les ressources dans les systèmes répartis. Ce modèle assure que les messages ou les données sont traités dans l'ordre dans lequel ils sont reçus, ce qui est crucial pour maintenir l'intégrité et la cohérence des données dans un environnement distribué.

2.5.1 Notions de qualité de service

Les notions de qualité de service comprennent des paramètres tels que la bande passante, la latence, la disponibilité, et la fiabilité. Ces paramètres sont essentiels pour évaluer la performance d'un système distribué et déterminer si les exigences des utilisateurs sont satisfaites. En définissant des niveaux de service spécifiques, les systèmes peuvent garantir une QoS adéquate, permettant aux utilisateurs de bénéficier de performances prévisibles et fiables.

2.5.2 Modèle FIFO

Le modèle FIFO assure que les messages ou les données sont traités dans l'ordre dans lequel ils arrivent. Ce modèle est particulièrement important dans les systèmes répartis, car il garantit que les données sont traitées de manière séquentielle et cohérente, évitant ainsi les incohérences qui pourraient survenir si les messages étaient traités dans un ordre aléatoire. Le modèle FIFO est largement utilisé dans la gestion des files d'attente, la communication interprocessus, et d'autres applications nécessitant une gestion ordonnée des données.

2.5.3 Mécanismes de QoS

Les mécanismes de qualité de service dans les systèmes répartis comprennent des stratégies de gestion des ressources, de contrôle de la bande passante, et d'optimisation des performances. Cela peut inclure l'utilisation de files d'attente, de tampons, et d'algorithmes de gestion de la congestion pour assurer une distribution efficace des ressources. Les mécanismes de QoS peuvent également impliquer des politiques de priorité pour garantir que les messages critiques sont traités en premier, assurant ainsi une expérience utilisateur optimale dans les systèmes répartis.

Algorithm 4 Algorithme QoS avec un réseau FIFO

Variables :
$Q \leftarrow$ File FIFO vide ▷ File des messages envoyés
$P \leftarrow$ Processus expéditeur, $R \leftarrow$ Processus récepteur
$M_i \leftarrow$ Message, $timeout \leftarrow T_{\max}$ ▷ Durée maximale d'attente
Processus P (Expéditeur)
for chaque M_i à envoyer **do**
    Ajouter M_i à Q ▷ Ajout du message à la file FIFO
    Envoyer_message(M_i)
end for
Processus R (Récepteur)
while Réception de message **do**
    Lire_message(M_i)
    M_i est ajouté à la file dans l'ordre de réception
    if M_i est le prochain message attendu (FIFO) **then**
        Traiter_message(M_i) ▷ Message traité dans l'ordre FIFO
    else
        Démarrer_timer($timeout$) ▷ Ajout du timer pour le message attendu
        while true do
            if Message attendu reçu avant expiration du timer **then**
                Ajouter M_i à la file et Traiter_message(M_i)
                break
            else if Timer_expiré() **then**
                Demander_réémission(M_i) ▷ Demande de retransmission en cas de timeout
                Démarrer_timer($timeout$) ▷ Redémarrer le timer après demande
            end if
        end while
    end if
end while
QoS :
Le réseau garantit que les messages sont reçus dans l'ordre d'envoi ($FIFO$). Un mécanisme de timeout est ajouté pour gérer les retards, avec une retransmission demandée en cas de dépassement de délai.

2.5.4 Métriques et critères de qualité

Les métriques de qualité de service comprennent la latence, le débit, la gigue et la perte de données. La latence fait référence au temps de propagation des données à travers le réseau, tandis que le débit mesure la quantité de données pouvant être transmises sur une période donnée. La gigue fait référence à la variation de la latence, tandis que la perte de données indique le pourcentage de paquets qui n'atteignent pas leur destination.

2.5.5 Garanties de qualité de service

Les garanties de qualité de service impliquent des accords de niveau de service (SLA) entre les fournisseurs de réseau et les utilisateurs. Ces accords spécifient les niveaux de performance attendus en termes de latence, de débit et de fiabilité. Les fournisseurs de réseau doivent mettre en place des mécanismes de surveillance et de contrôle pour garantir le respect des SLA, et les utilisateurs ont souvent la possibilité de demander des compensations en cas de non-respect des engagements de qualité de service.

2.6 Conclusion

Les systèmes répartis sont essentiels dans le paysage informatique moderne, et comprendre les concepts clés de communication, de contrôle de flux, et de qualité de service est crucial pour concevoir et gérer efficacement ces systèmes. En intégrant ces concepts, les développeurs et les ingénieurs peuvent créer des applications plus performantes et fiables, répondant ainsi aux besoins croissants des utilisateurs et des organisations.

Chapitre 3

Le Temps

3.1 Introduction

Les définitions et l'importance du temps dans les systèmes répartis permettent de clarifier les notions essentielles telles que la cohérence, la synchronisation et la précision temporelle. Comprendre les concepts de temps est crucial pour garantir la fiabilité et la stabilité des systèmes répartis. L'importance du temps dans ce contexte réside dans sa capacité à affecter directement le bon fonctionnement des applications, ainsi que la cohérence des données et la gestion des événements.

3.2 Temps horloge

Les horloges logiques dans les systèmes répartis sont des dispositifs utilisés pour coordonner les événements. Elles sont basées sur des valeurs d'horloge attribuées à chaque événement dans le système. Ces horloges logiques permettent de maintenir un ordre partiel sur les événements en déterminant si un événement a eu lieu avant ou après un autre. En utilisant des mécanismes de communication entre les nœuds du système, les horloges logiques garantissent la cohérence des événements dans un environnement distribué, malgré la variation des horloges physiques entre les différents nœuds.

3.2.1 Définition et fonctionnement des horloges logiques

Les horloges logiques sont des outils essentiels pour la synchronisation du temps dans les systèmes répartis. Elles sont basées sur des marqueurs logiques qui permettent de construire un ordre partiel entre les événements. Leur fonctionnement repose sur la transmission de messages entre les différents nœuds du système, et sur l'attribution de valeurs aux marqueurs logiques pour ordonner les événements. Les horloges logiques permettent ainsi de reconstruire un ordre partiel des événements dans un système distribué, en déterminant la causalité entre ces événements.

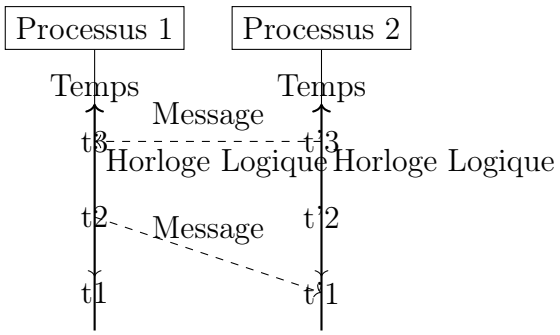


FIGURE 3.1 – Schéma des Horloges Logiques dans les Systèmes Répartis

Algorithm 5 Algorithme de Fonctionnement des Horloges Logiques de Lamport

Input : Ensemble des processus $P = \{P_1, P_2, \ldots, P_n\}$ dans le système réparti
Output : Maintien d'un ordre partiel des événements via horloges logiques
    Processus P_i
Initialise l'horloge logique $C_i \leftarrow 0$
    Événement local e dans P_i
Incrémente $C_i \leftarrow C_i + 1$
Associe $C_i(e)$ comme timestamp de e
    Message envoyé de P_i à P_j
Incrémente $C_i \leftarrow C_i + 1$
Inclut C_i dans le message
if Processus P_j reçoit un message **then**
    $C_j \leftarrow \max(C_j, C_i) + 1$
    Associe C_j comme timestamp pour l'événement de réception
end if

3.3 Temps environnement

L'environnement dans lequel les systèmes répartis opèrent peut avoir un impact significatif sur la gestion du temps. Les variations de température, d'humidité et d'autres facteurs environnementaux peuvent affecter la précision des horloges, ce qui peut entraîner des erreurs de synchronisation. Par conséquent, il est essentiel de prendre en compte l'environnement lors de la conception et de la mise en œuvre des systèmes répartis pour assurer une gestion précise du temps.

3.3.1 Impact de l'environnement sur la gestion du temps dans les systèmes répartis

L'impact de l'environnement sur la gestion du temps dans les systèmes répartis est crucial car les conditions environnementales peuvent perturber la transmission et la réception des signaux horaires, ce qui peut entraîner des incohérences dans la gestion du

temps. Par exemple, des fluctuations de tension électrique dues à des conditions environnementales instables peuvent affecter la fréquence des horloges, entraînant des erreurs de synchronisation. Il est donc essentiel de tenir compte de ces facteurs environnementaux lors de la conception et de la mise en œuvre des systèmes répartis pour garantir une gestion précise du temps malgré les variations environnementales.

3.4 Environnement synchrone

Dans les systèmes répartis, un environnement synchrone se caractérise par des processus qui progressent à un rythme uniforme et coordonné. Cela permet une gestion plus précise du temps et une synchronisation efficace entre les différentes entités du système. Les applications des environnements synchrones sont nombreuses, notamment dans le domaine des communications en temps réel, des transactions financières et des systèmes de contrôle industriels où la cohérence temporelle est cruciale pour garantir des opérations fiables et sécurisées.

3.4.1 Caractéristiques et applications des environnements synchrones

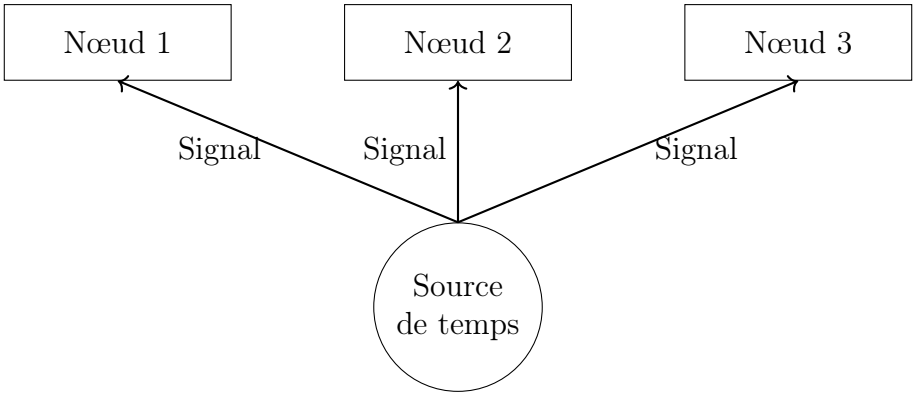
Les environnements synchrones se distinguent par une synchronisation stricte des activités, permettant d'éviter les problèmes liés à la latence et à la concurrence. Cela offre des avantages pour les applications nécessitant un haut degré de fiabilité et de précision temporelle, comme les simulations scientifiques, les systèmes d'imagerie médicale et les réseaux de capteurs. En outre, les environnements synchrones sont essentiels pour garantir la cohérence des données distribuées et la coordination des tâches dans les environnements critiques et exigeants en termes de temps, tels que les systèmes de contrôle aérien et les applications de réalité virtuelle.

3.5 Temps physique

Le temps physique fait référence au temps mesuré par une horloge réelle, basée sur des phénomènes physiques tels que les oscillations atomiques. Contrairement au temps logique, le temps physique est absolu et ne dépend pas de la perception ou de la logique des processus dans un système distribué. Il s'agit d'une mesure objective, indépendante de tout facteur externe et essentielle pour la synchronisation des événements dans un environnement réparti.

3.5.1 Différences entre temps physique et temps logique dans les systèmes répartis

Les différences entre le temps physique et le temps logique dans les systèmes répartis résident dans leur nature et leur utilisation. Le temps physique est basé sur des événements mesurables dans le monde réel, tandis que le temps logique dépend de la perception et de l'interprétation des événements par les processus distribués. Cette distinction est cruciale pour la gestion des transactions et la cohérence des données, car le temps physique



Ce schéma illustre la synchronisation des horloges physiques dans un système réparti. La source de temps émet des signaux vers chaque nœud afin de garantir une synchronisation temporelle fiable et cohérente entre les différents composants du système.

FIGURE 3.2 – Schéma de la synchronisation des horloges physiques dans un système réparti

garantit un repère absolu pour l'ordonnancement des opérations, tandis que le temps logique offre une vision subjective basée sur les interactions entre les processus.

3.6 Horloges physiques synchrones

Les horloges physiques synchrones dans les systèmes répartis jouent un rôle crucial dans la synchronisation des événements. Elles permettent d'obtenir une référence temporelle commune et précise pour l'ensemble des composants du système. Cette synchronisation est essentielle pour garantir la cohérence des données et l'ordonnancement des opérations. Les horloges physiques synchrones sont utilisées pour éviter les incohérences temporelles entre les différents nœuds du système, assurant ainsi un fonctionnement fiable

Algorithm 6 Algorithme de Synchronisation des Horloges Physiques

Require: Ensemble des nœuds $N = \{N_1, N_2, \ldots, N_k\}$, source de temps T
Ensure: Horloges synchronisées $H = \{H_1, H_2, \ldots, H_k\}$
Étape 1 : Obtenir l'heure locale de chaque nœud nœud $N_i \in N$
Obtenir l'heure locale actuelle : $H_i \leftarrow$ heure locale de N_i

Étape 2 : Obtenir l'heure de référence de la source de temps
$T \leftarrow$ heure universelle fournie par la source de temps
Étape 3 : Calculer et appliquer les corrections nœud $N_i \in N$
Calculer le décalage : $\Delta_i \leftarrow T - H_i$ ▷ Différence entre l'heure locale et la source
Synchroniser l'horloge : $H_i \leftarrow H_i + \Delta_i$ ▷ Met à jour l'heure de N_i

Retourner : Les horloges synchronisées H

et prévisible.

3.6.1 Technologies et protocoles utilisés pour synchroniser les horloges physiques

La synchronisation des horloges physiques dans les systèmes répartis repose sur l'utilisation de différentes technologies et protocoles. Parmi les technologies couramment utilisées, on trouve le protocole NTP (Network Time Protocol) qui permet de synchroniser les horloges en se basant sur des sources de temps fiables. D'autres technologies telles que PTP (Precision Time Protocol) sont également utilisées pour assurer une précision temporelle élevée. Ces protocoles permettent de maintenir la synchronisation des horloges physiques à travers le réseau, en prenant en compte les délais de transmission et les variations de la latence.

3.7 Conclusion

Le concept de temps dans les systèmes répartis est fondamental pour la synchronisation, la coordination et la fiabilité des processus distribués. Ce chapitre a exploré les divers aspects du temps, notamment les horloges logiques et physiques, ainsi que l'impact de l'environnement et les caractéristiques des environnements synchrones. La compréhension de ces éléments est cruciale pour concevoir des systèmes capables de maintenir la cohérence temporelle et de garantir des interactions harmonieuses entre les composants.

Les horloges logiques permettent de maintenir un ordre relatif entre les événements dans des systèmes où la synchronisation absolue est difficile à atteindre. Les horloges physiques, quant à elles, fournissent une base de référence temporelle plus précise, bien que souvent sujette aux variations environnementales. Les environnements synchrones facilitent cette synchronisation grâce à des processus évoluant de manière coordonnée, ce qui est essentiel dans des contextes où la précision temporelle est cruciale.

En conclusion, le temps et la gestion temporelle sont des aspects vitaux dans les systèmes répartis. En maîtrisant les concepts et technologies associés, il devient possible de développer des applications distribuées plus robustes et performantes, répondant aux exigences de précision et de fiabilité nécessaires dans les environnements critiques et temps réel.

Chapitre 4

Les algorithmes de Concurrence

4.1 Introduction

Cette section se concentre sur la définition précise des termes clés tels que la concurrence, la cohérence et les systèmes répartis. Elle vise à établir une base solide en clarifiant les concepts fondamentaux qui seront étudiés en profondeur par la suite. En outre, on fournit une vue d'ensemble des principaux défis et enjeux associés à ces concepts dans le contexte des systèmes répartis.

4.1.1 Importance de la Concurrence et de la Cohérence

L'importance de la concurrence et de la cohérence dans les systèmes répartis réside dans leur impact direct sur la performance, la fiabilité et la tolérance aux pannes de ces systèmes. Il est crucial de comprendre en profondeur les mécanismes de concurrence et les techniques de maintien de la cohérence des données pour garantir un fonctionnement fiable et efficace des systèmes répartis, ce qui justifie l'analyse approfondie des algorithmes associés à ces domaines.

4.2 Algorithmes de Concurrence de Lamport

Les algorithmes de concurrence de Lamport sont largement utilisés dans les systèmes répartis pour gérer les problèmes de concurrence. L'algorithme de Lamport repose sur le concept d'estampilles logiques pour ordonner les événements dans un système distribué. En utilisant des estampilles logiques, cet algorithme permet de déterminer l'ordre dans lequel les événements se produisent, ce qui est essentiel pour garantir la cohérence des données. L'application efficace de l'algorithme de Lamport nécessite une compréhension approfondie de ses principes fondamentaux et de son fonctionnement.

4.2.1 Fondements et Principes de l'Algorithme de Lamport

Les fondements de l'algorithme de Lamport reposent sur la notion d'horloge logique, qui attribue une estampille logique à chaque événement dans un système distribué. Ces estampilles logiques sont utilisées pour ordonner les événements et résoudre les problèmes de concurrence. Le principe clé de l'algorithme de Lamport est de garantir que si l'événement A causé l'événement B, alors l'estampille logique de A est inférieure à celle de

B. Cette approche permet de maintenir la cohérence des données et d'éviter les conflits dans les systèmes répartis.

4.2.2 Mécanismes de Contrôle de la Concurrence

Les mécanismes de contrôle de la concurrence dans l'algorithme de Lamport impliquent l'utilisation judicieuse des estampilles logiques pour gérer les opérations concurrentes. En attribuant des estampilles logiques adéquates, les systèmes distribués peuvent garantir un ordre cohérent des événements et éviter les incohérences. De plus, les mécanismes de contrôle de la concurrence visent à minimiser les conflits et à optimiser les performances des systèmes répartis en réduisant les temps d'attente et les erreurs de concurrence.

Algorithm 7 Algorithme de Lamport

Entrée : Nœuds $P = \{P_1, P_2, \ldots, P_k\}$

Sortie : Horloge logique C_i pour chaque processus P_i
    chaque processus P_i
$C_i \leftarrow 0$ Horloge logique initialisée
    true
$C_i \leftarrow C_i + 1$ Incrémente l'horloge logique
Exécutez la section critique
$C_i \leftarrow C_i + 1$ Incrémente à nouveau l'horloge
    chaque P_j voisin
Envoyez un message avec horloge C_i à P_j
$E_i \leftarrow$ Libre
Attendez les réponses des messages
    reçoit un message de P_j avec horloge C_j
$C_i \leftarrow \max(C_i, C_j) + 1$ Mise à jour de l'horloge
$E_i \leftarrow$ Exécution

4.3 Algorithmes de Concurrence de Ricart et Agrawala

Dans cette section, nous nous pencherons sur les algorithmes de concurrence de Ricart et Agrawala, qui visent à garantir la sécurité et l'efficacité des systèmes répartis. En particulier, nous examinerons en détail l'algorithme de Ricart et Agrawala, en mettant en lumière sa structure, ses principes de fonctionnement et son impact sur la concurrence. Il s'agit d'une opportunité de mieux comprendre comment cet algorithme aborde les défis de la concurrence dans les systèmes répartis, en mettant l'accent sur sa pertinence et son application pratique.

4.3.1 Compréhension de l'Algorithme de Ricart et Agrawala

Dans cette partie, nous approfondirons notre compréhension de l'algorithme de Ricart et Agrawala, en analysant ses mécanismes clés, ses avantages et ses potentielles limites. Nous examinerons également son fonctionnement dans des scénarios spécifiques, afin de

mieux saisir son impact sur la concurrence et la cohérence dans les systèmes répartis. Notre objectif est de fournir une vision approfondie de cet algorithme, en mettant en évidence ses implications pratiques et ses applications potentielles.

Algorithm 8 Algorithme de Ricart et Agrawala

Entrée : Nœuds $P = \{P_1, P_2, \ldots, P_k\}$

Sortie : État de chaque processus E_i (Libre, Attente, Exécution)
    chaque processus P_i
$C_i \leftarrow 0$ Horloge logique initialisée
$E_i \leftarrow$ Libre État initial
$Q_i \leftarrow \emptyset$ File d'attente vide
    true $E_i =$ Libre
$C_i \leftarrow C_i + 1$ Incrémente l'horloge logique
Exécutez la section critique
$E_i \leftarrow$ Exécution
    chaque P_j voisin
Envoyez une demande à P_j
$E_i \leftarrow$ Attente
Attendez les réponses des voisins
$E_i \leftarrow$ Libre $E_i =$ Attente
Attendez la réponse des voisins

4.3.2 Comparaison avec d'Autres Approches

Dans cette section, nous comparerons l'algorithme de Ricart et Agrawala avec d'autres approches de gestion de la concurrence dans les systèmes répartis, telles que : - L'algorithme du jeton tournant (token-based), - Les approches basées sur les horloges logiques de Lamport.

Nous mettrons en évidence les forces et les faiblesses de l'algorithme de Ricart et Agrawala par rapport à ces solutions en examinant leurs performances, leur complexité et leur facilité de mise en œuvre.

Algorithme de Ricart et Agrawala

- **Principe :** Cet algorithme utilise un système de requêtes et d'acquittements pour accéder à la section critique. Un processus doit envoyer une requête à tous les autres processus et attendre leurs acquittements avant d'entrer dans la section critique. - **Forces :** - Évite les défaillances liées au jeton perdu (comme dans le jeton tournant). - N'assume pas de structure logique particulière, ce qui le rend adaptable à divers réseaux. - **Faiblesses :** - Génère un grand nombre de messages (2 * (N - 1) messages pour une section critique). - Sensible aux défaillances des processus, car un processus inactif peut bloquer tout le système.

Comparaison avec l'Algorithme du Jeton Tournant

- **Principe :** Le jeton tournant repose sur la possession d'un jeton unique. Seul le processus possédant le jeton peut accéder à la section critique. - **Avantages du jeton

tournant :** - Réduction significative du trafic réseau : un seul message est nécessaire pour transférer le jeton. - Faible latence dans des systèmes bien configurés. - **Inconvénients par rapport à Ricart et Agrawala :** - Vulnérable à la perte du jeton, nécessitant des mécanismes de régénération. - Peut entraîner une attente prolongée si le jeton est bloqué chez un processus inactif.

Comparaison avec les Horloges Logiques de Lamport

- **Principe :** Les horloges logiques garantissent l'ordre causal des événements, mais ne fournissent pas de mécanisme explicite pour gérer l'accès à la section critique. - **Avantages des horloges de Lamport :** - Simplicité conceptuelle et faible complexité de mise en œuvre. - Utile dans des systèmes où l'ordre des événements est critique mais où les conflits d'accès sont rares. - **Inconvénients par rapport à Ricart et Agrawala :** - Ne garantit pas l'exclusion mutuelle. - Requiert une combinaison avec d'autres mécanismes pour la gestion de la concurrence.

array

Analyse Comparative

Critères	**Ricart et Agrawala**	**Jeton Tournant**	**Horloges de Lamport**
Messages échangés	$2(N-1)$	1 par section critique	Variable, selon l'usage
Tolérance aux pannes	Faible (blocage possible)	Moyenne (perte du jeton)	Élevée (systèmes asynchrones)
Complexité	Moyenne	Faible	Faible
Scalabilité	Moyenne	Élevée	Moyenne

TABLE 4.1 – Comparaison entre Ricart et Agrawala, le jeton tournant, et les horloges de Lamport.

Conclusion : L'algorithme de Ricart et Agrawala se distingue par sa simplicité et son absence de dépendance à un jeton unique. Cependant, sa complexité en termes de messages échangés et sa faible tolérance aux pannes peuvent limiter son utilisation dans des systèmes distribués à grande échelle. Les algorithmes comme le jeton tournant sont plus efficaces pour réduire le trafic réseau, mais introduisent des vulnérabilités spécifiques. En revanche, les horloges logiques sont mieux adaptées à des scénarios où l'ordre des événements est prioritaire, mais nécessitent des mécanismes supplémentaires pour l'exclusion mutuelle.

4.4 Algorithmes de Concurrence de Carvalho et Roucairol

L'algorithme de concurrence de Carvalho et Roucairol est basé sur le concept de graphe de préférence, où les processus ont des priorités pour accéder à des ressources partagées. Cette approche vise à réduire les attentes et à optimiser l'utilisation des ressources. En outre, l'algorithme garantit l'absence de famine, car chaque processus a la

possibilité d'accéder à la ressource partagée. Le mécanisme de coordination repose sur des communications asynchrones, ce qui rend l'algorithme adapté aux systèmes répartis à grande échelle et présentant des délais variables. En outre, l'algorithme est robuste face aux pannes de processus, assurant ainsi une haute disponibilité des ressources.

4.4.1 Architecture et Fonctionnement de l'Algorithme de Carvalho et Roucairol

L'architecture de l'algorithme de Carvalho et Roucairol repose sur un mécanisme de demande et d'octroi de ressources basé sur les priorités. Chaque processus maintient un vecteur de priorité qui est utilisé pour déterminer l'ordre d'accès aux ressources partagées. Lorsqu'un processus souhaite accéder à une ressource, il envoie une demande aux autres processus avec une priorité plus élevée. Une fois la ressource accordée, le processus peut l'utiliser en toute sécurité, sous réserve de libérer la ressource en temps opportun pour éviter les blocages. Cette approche garantit la justesse et l'efficacité de l'accès concurrent aux ressources dans les systèmes répartis.

Algorithm 9 Algorithme de Carvalho et Roucairol

Require: Ensemble des nœuds $P = \{P_1, P_2, \ldots, P_k\}$
Ensure: État de chaque processus $E_i \in \{\text{Libre}, \text{Attente}, \text{Exécution}\}$
for all processus $P_i \in P$ **do**
    $C_i \leftarrow 0$ ▷ Initialiser l'horloge logique de P_i
    $E_i \leftarrow$ Libre ▷ État initial de P_i
end for
while true **do**
    if $E_i =$ Libre **then**
        $C_i \leftarrow C_i + 1$ ▷ Incrémenter l'horloge logique locale
        $E_i \leftarrow$ Attente ▷ Passer à l'état d'attente pour accéder à la section critique
        for all P_j voisin de P_i **do**
            Envoyer une demande à P_j avec horloge C_i
        end for
        Attendre les réponses de tous les voisins ▷ Synchronisation pour accéder à la section critique
        $E_i \leftarrow$ Exécution ▷ Accès à la section critique
        Exécuter la section critique
        $E_i \leftarrow$ Libre ▷ Retour à l'état Libre
        Informer les voisins de la libération
    else if $E_i =$ Attente **then**
        Attendre la réception des autorisations des voisins
    end if
end while

4.4.2 Applications et Limitations

Les applications de l'algorithme de concurrence de Carvalho et Roucairol incluent les systèmes de gestion de bases de données distribuées, les réseaux de capteurs sans fil et les environnements IoT. L'algorithme offre des performances élevées en termes d'utilisation

des ressources et de prévention des conflits. Cependant, certaines limitations subsistent, notamment en ce qui concerne la complexité de la gestion des priorités dans des environnements dynamiques, ainsi que la surcharge de communication lors de l'attribution de ressources. Des études continues sont nécessaires pour adapter l'algorithme aux exigences évolutives des systèmes répartis modernes.

4.5 Études de Cas et Implémentations Réelles

Cette section se concentre sur l'analyse d'études de cas et d'implémentations réelles des algorithmes de concurrence dans les systèmes répartis. Nous examinerons des exemples concrets d'application de ces algorithmes dans des environnements pratiques, mettant en lumière les défis rencontrés, les solutions apportées et les leçons apprises. L'objectif est de fournir une compréhension approfondie de la mise en œuvre réelle de ces algorithmes et de leur incidence sur les performances et la cohérence des systèmes répartis.

4.5.1 Exemples d'Applications Pratiques

Dans cette sous-section, nous présenterons des exemples concrets d'applications pratiques des algorithmes de concurrence dans les systèmes répartis. Nous analyserons comment ces algorithmes sont utilisés pour gérer la concurrence et assurer la cohérence des données dans des scénarios réels, tels que les bases de données distribuées, les systèmes de gestion de fichiers distribués et les environnements de cloud computing. Nous mettrons en lumière les avantages et les défis associés à chaque application, ainsi que les leçons apprises pour une mise en œuvre réussie.

4.5.2 Déploiement et Intégration dans des Environnements Réels

Dans cette sous-section, nous examinerons le déploiement et l'intégration des algorithmes de concurrence dans des environnements réels. Nous aborderons les défis liés à l'implémentation de ces algorithmes dans des systèmes existants, ainsi que les stratégies pour assurer une intégration efficace sans perturber les opérations en cours. Nous mettrons en évidence les bonnes pratiques, les outils et les techniques utilisés pour déployer avec succès ces algorithmes dans des environnements réels, en tenant compte des contraintes de performance, de fiabilité et de disponibilité.

4.6 Comparaison et Analyse des Performances

Lors de la comparaison des performances des algorithmes de concurrence dans les systèmes répartis, plusieurs paramètres sont pris en compte, tels que le temps de réponse, l'efficacité énergétique, la scalabilité et la tolérance aux pannes. Ces critères permettent d'évaluer la capacité de chaque algorithme à gérer la concurrence et à maintenir la cohérence des données dans des environnements distribués. Une analyse approfondie des performances met en lumière les avantages et les inconvénients de chaque approche, offrant ainsi un aperçu complet de leur fonctionnement dans des conditions réelles.

4.6.1 Critères d'Évaluation

Les critères d'évaluation des algorithmes de concurrence incluent la justesse, la complétude, la performance, la simplicité et la modularité. Ces critères permettent d'analyser la capacité des algorithmes à garantir la cohérence des données, à éviter les incohérences et à maintenir des performances élevées. En évaluant ces critères, il est possible de déterminer la pertinence de chaque algorithme dans des contextes spécifiques, offrant ainsi des indications précieuses pour leur mise en œuvre dans des systèmes répartis.

4.6.2 Scénarios d'Utilisation et Résultats

Les scénarios d'utilisation des algorithmes de concurrence dans des systèmes répartis varient en fonction des besoins spécifiques, tels que la lecture et l'écriture de données, la gestion des transactions et la coordination des processus distribués. En analysant les résultats obtenus dans différents scénarios, il est possible d'identifier les forces et les faiblesses de chaque algorithme, ainsi que de déterminer le meilleur choix pour des applications particulières. Les résultats de ces analyses fournissent des indications précieuses pour la sélection et l'implémentation d'algorithmes de concurrence dans des systèmes répartis.

4.7 Analyse de la complexité des algorithmes de concurrence dans les systèmes répartis

L'analyse de la complexité des algorithmes dans les systèmes répartis, notamment ceux de concurrence, est essentielle pour évaluer leur efficacité et leur résilience. Cette analyse s'étend au-delà de la simple mesure du temps d'exécution et prend en compte trois aspects principaux : le temps d'exécution, les messages échangés et la tolérance aux pannes.

Complexité en termes de temps

L'efficacité temporelle d'un algorithme de concurrence dans un système réparti dépend de plusieurs facteurs, dont la synchronisation des processus et la gestion des ressources partagées. Les algorithmes doivent être conçus pour minimiser le temps d'attente et maximiser l'utilisation des ressources sans provoquer de conflits. La complexité temporelle se mesure souvent par le nombre de cycles nécessaires pour atteindre un état stable, ce qui peut être influencé par la latence de communication entre les différents nœuds du système et par la gestion des verrous ou des sémaphores.

Complexité en termes de messages échangés

Les systèmes répartis impliquent souvent une communication constante entre les processus. L'efficacité de cette communication est un critère clé dans l'évaluation des algorithmes de concurrence. L'analyse de la complexité en termes de messages échangés se concentre sur le nombre de messages nécessaires pour que les processus synchronisent leurs actions ou pour qu'ils partagent des informations. Les algorithmes doivent être optimisés pour réduire le nombre de messages tout en garantissant une communication fiable et rapide entre les nœuds.

Tolérance aux pannes

La tolérance aux pannes est un aspect crucial des algorithmes de concurrence dans les systèmes répartis. Un algorithme tolérant aux pannes doit être capable de continuer à fonctionner même en cas de défaillance d'un ou plusieurs composants du système. Cela inclut la gestion des pannes de nœuds, la récupération d'état après une panne et l'intégrité des données partagées. L'analyse de la tolérance aux pannes se base sur la capacité de l'algorithme à maintenir la cohérence des données tout en minimisant les interruptions dans les processus concurrents.

En conclusion, l'intégration de ces analyses dans l'étude des algorithmes de concurrence dans les systèmes répartis permet de mieux comprendre leur efficacité, leur résilience et leur adaptabilité dans des environnements dynamiques et distribués. Ces considérations doivent être prises en compte lors de la conception des algorithmes pour s'assurer qu'ils répondent aux exigences de performance et de fiabilité.

4.8 Conclusion et Synthese

En conclusion, cette analyse approfondie des algorithmes de concurrence dans les systèmes répartis met en évidence l'importance de ces mécanismes pour garantir la cohérence des données et assurer des performances fiables. Les algorithmes de Lamport, Ricart et Agrawala, ainsi que Carvalho et Roucairol, offrent des solutions variées pour gérer la concurrence dans des environnements distribués, chacun avec ses propres avantages et limitations. Cette étude souligne l'importance de la recherche continue dans ce domaine pour faire face aux défis croissants posés par les applications distribuées et offre des perspectives prometteuses pour l'avenir.

4.8.1 Récapitulatif des Principaux Points

Pour récapituler, cette analyse a mis en lumière les principaux concepts et mécanismes des algorithmes de concurrence, y compris les approches de Lamport, Ricart et Agrawala, ainsi que Carvalho et Roucairol. Elle a également examiné en détail les critères d'évaluation des performances, les scénarios d'utilisation, ainsi que des exemples concrets d'applications pratiques et d'implémentations réelles. En outre, des défis non résolus et de nouvelles directions de recherche ont été identifiés, soulignant la complexité continue de ce domaine et l'importance de l'innovation future.

4.8.2 Implications et Importance des Algorithmes de Concurrence

Les implications et l'importance des algorithmes de concurrence dans les systèmes répartis sont significatives dans le contexte actuel des applications distribuées. La capacité à gérer efficacement la concurrence et à assurer la cohérence des données est cruciale pour garantir des performances fiables et éviter les problèmes de sécurité. Les algorithmes de Lamport, Ricart et Agrawala, ainsi que Carvalho et Roucairol, offrent des solutions clés pour relever ces défis, tout en présentant des opportunités potentielles pour de futures innovations et développements dans le domaine des systèmes distribués.

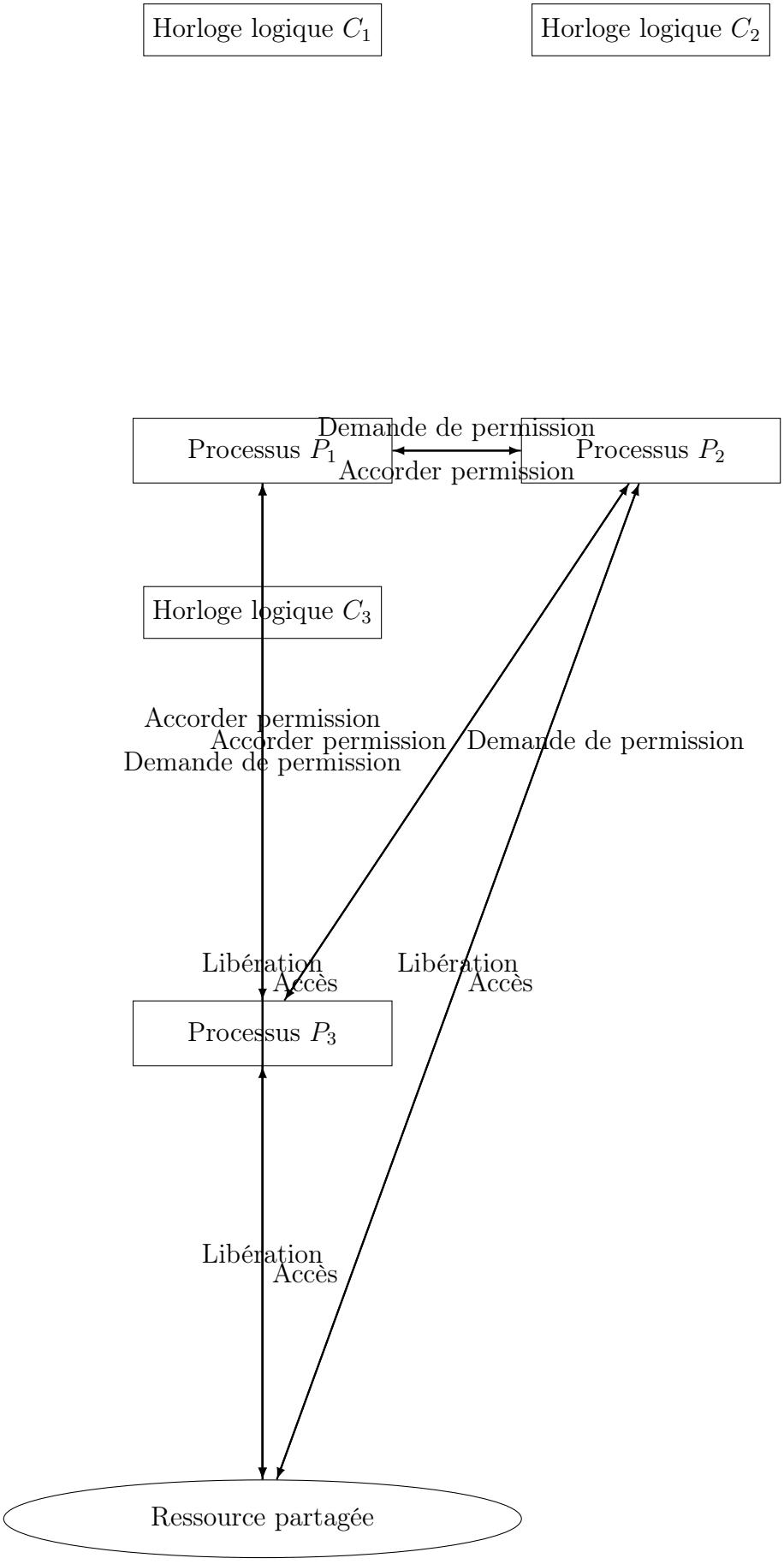


FIGURE 4.1 – Schéma de l'algorithme de Lamport.

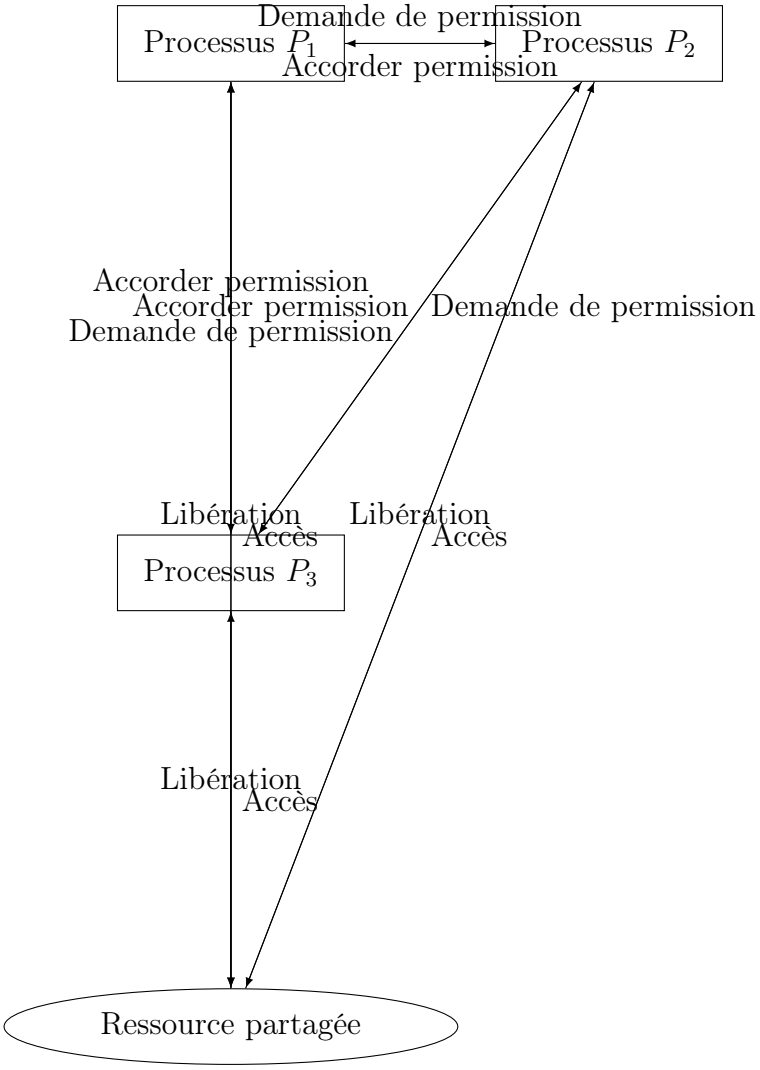


FIGURE 4.2 – Schéma de l'algorithme de Ricart et Agrawala.

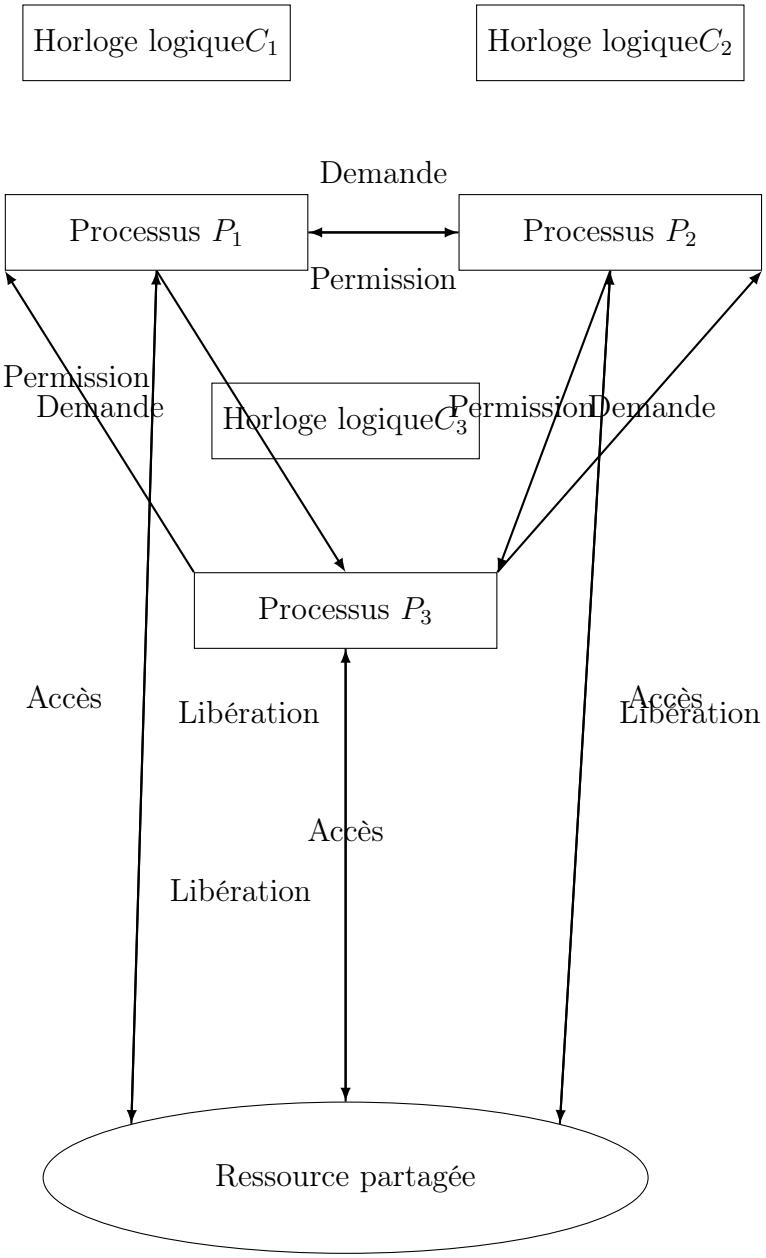


FIGURE 4.3 – Schéma de l'algorithme de Carvalho et Roucairol avec horloges logiques espacées.

Chapitre 5

L'observation

5.1 Introduction

Dans cette section, nous introduisons le concept d'observation dans les systèmes répartis, mettant en évidence son importance dans la surveillance et le suivi des applications distribuées. L'observation est essentielle pour garantir le bon fonctionnement et la fiabilité des systèmes répartis, permettant la détection précoce des éventuels problèmes et la prise de mesures correctives. Nous présentons également les principaux objectifs de l'observation dans ce contexte, en soulignant son rôle crucial pour maintenir la cohérence et la robustesse des applications réparties.

5.1.1 Définitions et Concepts Clés

Dans cette sous-section, nous abordons les définitions et concepts clés liés à l'observation dans les systèmes répartis. Nous clarifions la signification de l'observation, en mettant en lumière ses différents aspects tels que la collecte de données, l'analyse des métriques et la surveillance des comportements. De plus, nous identifions les principaux termes et concepts associés à l'observation des systèmes répartis, fournissant ainsi une base solide pour la compréhension approfondie de ce domaine crucial.

5.1.2 Importance de l'Observation dans les Systèmes Répartis

Cette section met en évidence l'importance de l'observation dans les systèmes répartis, soulignant son rôle central dans la garantie de la performance, de la fiabilité et de la disponibilité des applications réparties. Nous mettons en avant les bénéfices concrets de l'observation, tels que la détection proactive des défaillances, l'optimisation des ressources et la prise de décisions basées sur des données fiables. En comprenant pleinement l'importance de l'observation, les professionnels peuvent renforcer la qualité opérationnelle des systèmes répartis et offrir une meilleure expérience utilisateur.

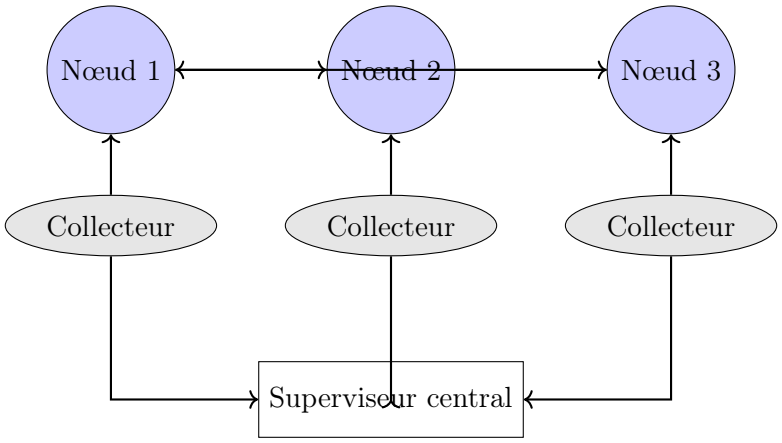
5.2 État d'une Application Répartie

L'état d'une application répartie fait référence à la condition actuelle et aux performances du système réparti. Comprendre cet état est essentiel pour assurer un fonctionnement optimal. Cela implique de surveiller les ressources système, la charge de travail, la

connectivité, les performances des applications, etc. L'observation de l'état d'une application répartie permet de détecter les goulots d'étranglement, les défaillances potentielles et d'optimiser les performances du système dans son ensemble.

5.2.1 Compréhension de l'État d'une Application Répartie

Comprendre l'état d'une application répartie nécessite une analyse approfondie de différents paramètres et métriques. Il s'agit d'observer le comportement des composants distribués, de surveiller les indicateurs de performance, de collecter des données sur la disponibilité des ressources et d'analyser les modèles de trafic. Cette compréhension permet de prendre des décisions éclairées pour l'optimisation et la gestion efficace de l'application répartie.



Surveillance et agrégation de l'état

FIGURE 5.1 – Schéma de l'état d'une application répartie avec collecte et supervision des données

5.2.2 Méthodes et Outils d'Observation de l'État

Les méthodes et outils d'observation de l'état d'une application répartie incluent la surveillance en temps réel, l'analyse des journaux, la collecte de données distribuées, l'utilisation de sondes et de capteurs pour recueillir des informations, ainsi que des techniques avancées telles que l'apprentissage automatique pour prédire les comportements futurs. Ces méthodes et outils sont essentiels pour évaluer de manière précise et détaillée l'état d'une application répartie.

5.3 Détection des Propriétés Stables et Paisibles

La détection des propriétés stables et paisibles dans les systèmes répartis est d'une importance capitale pour assurer le bon fonctionnement et la fiabilité des applications distribuées. En identifiant les propriétés stables, telles que la cohérence des données et la disponibilité du service, ainsi que les propriétés paisibles, comme la tolérance aux pannes et la réplication des ressources, on garantit une expérience utilisateur optimale. La détection de ces propriétés repose sur l'analyse en temps réel des comportements

du système et des données recueillies, permettant ainsi d'anticiper et de remédier aux éventuels dysfonctionnements.

5.3.1 Définitions et Caractéristiques des Propriétés Stables et Paisibles

Les propriétés stables dans un système réparti font référence à la capacité du système à maintenir un état cohérent malgré les événements qui surviennent, tandis que les propriétés paisibles concernent la capacité du système à résister aux pannes et aux erreurs. Ces propriétés garantissent la résilience et la fiabilité de l'application distribuée. Les caractéristiques des propriétés stables et paisibles incluent la redondance des données, la gestion des erreurs, la récupération après panne, et la communication asynchrone entre les composants du système.

5.3.2 Techniques de Détection des Propriétés

Pour détecter les propriétés stables et paisibles dans les systèmes répartis, diverses techniques sont utilisées, telles que l'analyse de l'état global du système, l'observation des transactions et des échanges de messages, l'identification des points de défaillance potentiels, et la mise en place de mécanismes de surveillance continue. Ces techniques permettent de garantir la résilience et la stabilité du système face aux conditions changeantes et aux éventuelles défaillances, assurant ainsi une expérience utilisateur sans heurts et un fonctionnement fiable de l'application distribuée.

Calcul et Définition de l'État Global

L'état global d'un système distribué est une vue complète des états locaux de tous les processus ainsi que des informations de communication entre eux. Dans un système distribué, chaque processus maintient un état local et échange des messages avec d'autres processus. L'état global est la combinaison de tous ces états locaux et des informations pertinentes sur les communications (par exemple, les messages envoyés ou reçus).

Mathematiquement, l'état global G d'un système distribué à un instant t peut être exprimé comme suit :

$$G(t) = \{\{S_1(t), S_2(t), \ldots, S_n(t)\}, \{M_1(t), M_2(t), \ldots, M_k(t)\}\}$$

où : - $S_i(t)$ représente l'état local du processus i à l'instant t, pour $i \in \{1, 2, \ldots, n\}$, avec n étant le nombre total de processus dans le système. - $M_j(t)$ représente les messages échangés entre les processus à l'instant t, pour $j \in \{1, 2, \ldots, k\}$, avec k étant le nombre total de messages échangés.

Ainsi, l'état global $G(t)$ combine les états locaux de tous les processus et les informations sur les messages, reflétant l'ensemble du système à un moment donné. Pour calculer cet état global, des techniques comme les *instantanés distribués* ou les *horloges logiques* peuvent être utilisées, où chaque processus enregistre son état local et les messages qu'il a reçus, tout en coordonnant cette information avec les autres processus.

Le calcul de l'état global dans un système distribué nécessite souvent une synchronisation des horloges ou des mécanismes de consensus afin de garantir que l'état global reflète fidèlement l'ordre causal des événements dans le système. Cela permet de détecter les anomalies, d'assurer la cohérence des données et de gérer les défaillances du système.

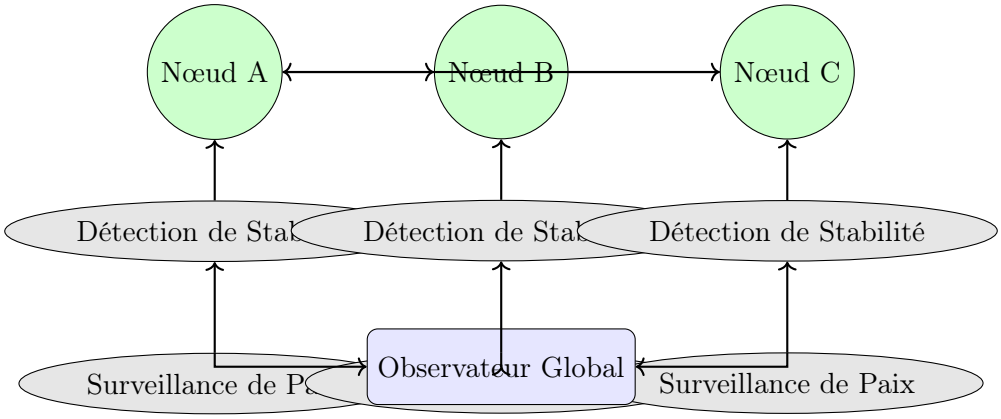


FIGURE 5.2 – Schéma de Détection des Propriétés Stables et Paisibles d'une Application Répartie

Pour la détection des propriétés stables et paisibles d'une application répartie, il est essentiel de surveiller les états qui ne changent pas avec le temps (stabilité) et les états où les ressources et communications sont équilibrées sans pic d'activité perturbateur (paix). Ces propriétés se repèrent souvent dans des systèmes qui atteignent un point d'équilibre après une période de transition. État Stable : Un état est stable si les propriétés de l'application, comme les valeurs des variables d'état ou l'état des processus, ne changent plus (ou restent dans une marge de fluctuation très faible) après une période d'évolution. État Paisible : Représente une faible charge sur les ressources et un faible nombre d'interruptions ou de messages. Ceci peut être observé par la surveillance de la latence, du débit réseau, et de l'utilisation CPU/RAM qui ne varient que faiblement.

5.4 Études de Cas et Applications Pratiques

Dans cette section, nous examinerons plusieurs exemples concrets d'applications réparties réelles où l'observation a joué un rôle crucial. Ces études de cas nous permettront de comprendre comment l'observation a été utilisée dans différents environnements et pour différents types d'applications, mettant en lumière la diversité des défis auxquels elle peut être confrontée et des solutions qui ont été développées.

5.4.1 Exemples d'Applications Réparties Réelles

Nous présenterons ici des exemples spécifiques d'applications réparties réelles dans divers domaines tels que les réseaux de capteurs, les applications web et les systèmes de traitement des données. Ces exemples illustreront comment l'observation a été mise en œuvre pour surveiller et gérer ces applications, ainsi que les leçons qui en ont été tirées pour l'amélioration des performances et de la fiabilité du système.

Systèmes Bancaires Distribués

Les systèmes bancaires modernes utilisent des bases de données réparties pour assurer la cohérence des transactions à travers différents pays et fuseaux horaires. Par exemple, le protocole de consensus est utilisé pour garantir que les transactions sont validées simultanément sur tous les serveurs. Les algorithmes de tolérance aux pannes jouent un

rôle crucial pour assurer une disponibilité continue.

Gestion de Réseaux de Capteurs

Les réseaux de capteurs distribués sont utilisés dans des domaines tels que la surveillance de l'environnement, où les capteurs collectent des données sur la température, l'humidité ou les niveaux de pollution. Ces données sont transmises à un serveur central à l'aide de protocoles de communication optimisés pour économiser l'énergie.

Applications IoT dans les Maisons Connectées

Dans les systèmes IoT (Internet des objets), les algorithmes distribués permettent de synchroniser les appareils domestiques connectés, tels que les thermostats intelligents ou les caméras de surveillance. Les horloges logiques et les protocoles de synchronisation garantissent une coordination précise entre les appareils.

Traitement des Données en Temps Réel

Des plateformes comme Apache Kafka et Apache Spark sont utilisées pour traiter de grandes quantités de données en temps réel dans des architectures distribuées. Par exemple, les entreprises de commerce électronique exploitent ces technologies pour analyser les comportements des utilisateurs et ajuster leurs offres instantanément.

Cloud Computing

Le déploiement de systèmes sur le cloud repose sur des algorithmes de répartition de charge et de virtualisation. Les services comme Amazon Web Services (AWS) ou Google Cloud utilisent des systèmes distribués pour garantir l'allocation dynamique des ressources en fonction de la demande des utilisateurs.

Systèmes de Contrôle dans l'Industrie 4.0

Les usines intelligentes exploitent des systèmes répartis pour surveiller et contrôler les machines en temps réel. Les algorithmes de coordination, combinés à des mécanismes d'observation, assurent une production continue et détectent les anomalies avant qu'elles ne perturbent les processus.

5.4.2 Résultats et Impacts de l'Observation dans les Applications Réparties

Nous examinerons les résultats obtenus grâce à l'observation dans les applications réparties réelles, en mettant en évidence les effets positifs sur la détection des problèmes, la gestion des incidents et l'optimisation des performances. Nous analyserons également les enseignements tirés de ces expériences et les bonnes pratiques qui peuvent être transposées et appliquées à d'autres systèmes distribués.

5.5 Défis et Perspectives Futures

Les défis dans l'observation des systèmes répartis incluent la garantie de la cohérence des données à travers des nœuds distants, la gestion des goulots d'étranglement de données et la résolution des problèmes de latence. De plus, il est essentiel de développer des solutions pour l'observation des systèmes répartis à grande échelle afin de garantir des performances optimales. Pour relever ces défis, des stratégies de surveillance avancées et des technologies de traitement des données en temps réel doivent être mises en place pour assurer une observation précise et fiable des systèmes distribués.

5.5.1 Défis Actuels dans l'Observation des Systèmes Répartis

Les défis actuels dans l'observation des systèmes répartis comprennent la complexité croissante des applications distribuées, la nécessité de surveiller les microservices et les conteneurs, ainsi que la garantie de la sécurité et de la confidentialité des données observées. De plus, il est crucial de développer des techniques de visualisation et d'analyse des données plus avancées pour une compréhension approfondie du comportement des systèmes distribués. La mise en place de mécanismes de détection d'anomalies en temps réel constitue également un défi majeur dans l'observation des systèmes répartis.

5.5.2 Nouvelles Directions de Recherche et Innovations

Les nouvelles directions de recherche et innovations dans l'observation des systèmes répartis se concentrent sur l'intégration de l'intelligence artificielle et de l'apprentissage automatique pour l'analyse prédictive des performances des systèmes distribués. De plus, l'exploration de nouvelles méthodes de collecte et de gestion des données distribuées, telles que l'utilisation de l'informatique en périphérie (edge computing), ouvre la voie à des innovations significatives dans le domaine de l'observation des systèmes répartis. En outre, l'adoption de normes et de protocoles de communication plus efficaces contribuera à améliorer la fiabilité et l'évolutivité des solutions d'observation des systèmes répartis.

5.6 Conclusion et Récapitulatif des Principaux Points

En conclusion, ce travail a examiné en détail l'observation dans les systèmes répartis, mettant en lumière l'importance de comprendre l'état d'une application répartie et la détection des propriétés stables et paisibles. Nous avons présenté des méthodes et outils d'observation de l'état, ainsi que des techniques de détection des propriétés, illustrant leur application à travers des études de cas réels. Les défis actuels dans ce domaine ont également été abordés, tout en mettant en avant les nouvelles directions de recherche et innovations. En récapitulant les principaux points, il est clair que l'observation dans les systèmes répartis joue un rôle crucial dans la garantie du bon fonctionnement des applications réparties, et que des efforts continus de recherche et développement sont nécessaires pour relever les défis à venir.

Chapitre 6

Les algorithmes d'élection

6.1 Introduction

Les définitions et caractéristiques des systèmes Les algorithmes d'élection sont essentiels dans les systèmes distribués car ils permettent de résoudre le problème de désignation d'un coordinateur ou d'un leader parmi un groupe de processus. Sans ces algorithmes, il serait difficile d'assurer la coordination et la gestion des tâches au sein du système. En effet, dans un système distribué, la désignation d'un leader est cruciale pour garantir la cohérence et la fiabilité des opérations. Ainsi, les algorithmes d'élection jouent un rôle fondamental dans l'organisation et le bon fonctionnement des systèmes distribués.

6.2 Algorithmes d'élection dans les systèmes distribués

Les algorithmes d'élection jouent un rôle crucial dans les systèmes distribués en permettant la sélection d'un coordinateur ou d'un leader parmi les différents processus. Leur importance réside dans le maintien de l'ordre et de la cohérence au sein du système. En effet, ces algorithmes garantissent qu'un seul processus soit élu comme leader, évitant ainsi les conflits et les incohérences. De plus, ils contribuent à la résilience du système en assurant la continuité des opérations même en cas de défaillance d'un processus. Enfin, ces algorithmes doivent être conçus pour minimiser la consommation de ressources et garantir des temps de réponse optimaux.

6.2.1 Algorithme de Chang et Roberts

L'algorithme de Chang et Roberts est l'un des premiers algorithmes d'élection développés pour les systèmes distribués. Il repose sur un réseau en anneau et utilise un protocole de transmission de messages pour élire un coordinateur parmi les nœuds du réseau. Ce processus se déroule en plusieurs étapes, chaque nœud envoyant des messages de candidature et de réponse pour déterminer le coordinateur. L'algorithme garantit l'élection d'un seul coordinateur, tout en minimisant la complexité et la consommation de bande passante. Sa conception robuste en fait un choix privilégié pour de nombreux cas d'utilisation dans lesquels la fiabilité et la rapidité de l'élection sont essentielles.

L'**algorithme de Chang et Roberts** est conçu pour élire un leader dans un réseau de processus organisés en anneau. Chaque processus (noté P_i) communique uniquement avec ses voisins immédiats.

6.2.2 Étapes de l'algorithme

1. **Initialisation** : Chaque processus a un identifiant unique. Les processus commencent par envoyer un message de candidature contenant leur identifiant à leur voisin de droite.
2. **Transmission des messages** : Lorsqu'un processus P_j reçoit un message de candidature contenant un identifiant :
   — Si l'identifiant du message est supérieur, P_j transfère le message à son voisin de droite.
   — Si l'identifiant du message est inférieur ou égal, P_j ignore le message.
3. **Élection du leader** : Lorsque le message de candidature revient à l'expéditeur, ce dernier sait que son identifiant est le plus élevé et envoie un message de notification à tous les autres processus, déclarant qu'il est le leader.
4. **Fin de l'élection** : Une fois qu'un processus a reçu le message de notification, il reconnaît que le leader a été élu.

6.2.3 Schéma de l'algorithme

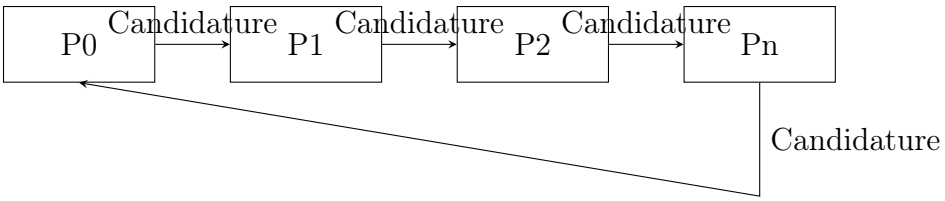


FIGURE 6.1 – Schéma de l'algorithme de Chang et Roberts

Chaque processus P_i envoie son identifiant au voisin de droite. Chaque processus compare son identifiant avec celui reçu et décide d'envoyer ou non le message. Une fois qu'un processus a parcouru tout l'anneau, il sait qu'il est le leader et en informe les autres.

6.2.4 Pseudo-code de l'algorithme

Voici un pseudo-code simplifié de l'algorithme de Chang et Roberts :

```
Procédure Election()
    identifiant <- Obtient l'identifiant du processus
    message <- identifiant

    // Envoie un message à droite
    Envoyer(message à P(i + 1))

    // Attendre de recevoir un message
    Réceptionner(message_reçu de P(i - 1))

    // Comparer les identifiants
    Si (message_reçu > identifiant) alors
        // Ignorer et passer le message
        Envoyer(message_reçu à P(i + 1))
    Sinon
```

```
        // Si c'est le plus grand identifiant, il est le leader
        Envoyer("Leader: " + identifiant à tous les processus)
    Fin Si
Fin Procédure
```

Cet algorithme est simple et efficace pour élire un leader dans des réseaux de processus interconnectés en anneau, tout en garantissant que seul un processus soit élu comme leader.

Algorithm 10 Algorithme de Chang et Roberts

initialiser : Chaque processus P_i a un identifiant unique ID_i
envoyer : $P_i \rightarrow P_{i+1}$: "Candidature : ID_i"
while true **do**
    $msg \leftarrow$ recevoir de P_{i-1}
    if $msg > ID_i$ **then**
        ignorer msg
        envoyer : $P_i \rightarrow P_{i+1}$: "msg"
    else if $msg < ID_i$ **then**
        envoyer : $P_i \rightarrow P_{i+1}$: "msg"
    else
        envoyer : $P_i \rightarrow$ tous : "Leader : ID_i"
        arrêter
    end if
end while

6.2.5 Algorithme de Franklin

L'algorithme de Franklin est une autre approche populaire pour l'élection de coordinateur dans les systèmes distribués. Contrairement à l'algorithme de Chang et Roberts, il s'appuie sur une topologie en arbre pour réaliser le processus d'élection. Chaque nœud envoie des messages de candidature à son parent dans l'arbre, et le parent retourne un message d'accusé de réception pour confirmer la coordination. Ce modèle hiérarchique simplifie le processus d'élection, réduisant ainsi la complexité et le nombre de messages échangés. L'algorithme de Franklin est particulièrement adapté aux systèmes distribués de grande taille, offrant une efficacité et une évolutivité accrues tout en maintenant la fiabilité de l'élection. L'**algorithme de Franklin** est conçu pour élire un leader dans un réseau de processus en utilisant des messages de vote. Chaque processus envoie son vote à d'autres processus, et le processus avec le plus grand nombre de votes devient le leader.

6.2.6 Étapes de l'algorithme

1. **Initialisation** : Chaque processus a un identifiant unique et commence par voter pour lui-même.
2. **Envoi de votes** : Chaque processus envoie son vote à un certain nombre d'autres processus. Les votes sont comptabilisés.
3. **Comptage des votes** : Chaque processus reçoit les votes des autres et les comptabilise.

4. **Élection du leader** : Le processus ayant le plus de votes est élu leader. En cas d'égalité, une règle de tie-breaker est appliquée (par exemple, choisir le processus avec l'identifiant le plus élevé).

6.2.7 Schéma de l'algorithme

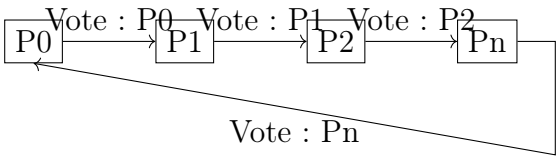


FIGURE 6.2 – Schéma de l'algorithme de Franklin

Chaque processus vote pour lui-même et envoie son vote à d'autres. Après avoir reçu tous les votes, chaque processus détermine qui a le plus de votes.

Voici l'algorithme formel de Franklin :

Algorithm 11 Algorithme de Franklin

initialiser : Chaque processus P_i a un identifiant unique ID_i et vote pour lui-même.
for chaque P_i **do**
    envoyer : $P_i \rightarrow P_j$: "Vote : ID_i"
end for
while votant **do**
    $votes \leftarrow$ recevoir de P_j
    compter : $votes[ID_i] \leftarrow votes[ID_i] + 1$
end while
déterminer : $leader \leftarrow \max(votes)$
if tie **then**
    appliquer : Règle de décomposition
end if

L'algorithme de Franklin permet une élection efficace dans un réseau de processus, où chaque processus peut voter pour lui-même et compter les votes des autres pour déterminer le leader.

6.3 Comparaison des Algorithmes d'Élection

La comparaison des algorithmes d'élection dans les systèmes distribués est essentielle pour choisir celui qui répond le mieux aux besoins d'une application donnée. Parmi les algorithmes étudiés, on retrouve l'algorithme de Chang et Roberts, l'algorithme de Franklin, et l'élection de type Bully. Ces algorithmes présentent des forces et des faiblesses en fonction de critères tels que la complexité, le nombre de messages échangés, la tolérance aux pannes, et la facilité de mise en œuvre.

6.3.1 Critères de Comparaison

— **Complexité en messages :** L'algorithme de Chang et Roberts utilise un nombre de messages proportionnel au nombre de processus $O(n)$. En revanche, l'algorithme

de Bully peut nécessiter jusqu'à $O(n^2)$ messages dans le pire des cas, ce qui peut entraîner une surcharge réseau dans de grands systèmes.
— **Tolérance aux pannes :** L'algorithme de Bully est robuste en cas de défaillance des processus non-leaders, mais il dépend fortement de la disponibilité du leader en cours. Franklin, quant à lui, assure une tolérance modérée grâce à sa gestion séquentielle des tours.
— **Temps de convergence :** Chang et Roberts convergent rapidement avec un minimum d'attente, tandis que Franklin et Bully peuvent nécessiter plus de temps selon les conditions du réseau.
— **Simplicité de mise en œuvre :** Chang et Roberts sont relativement simples à mettre en œuvre grâce à leur structure en anneau. Bully, en revanche, nécessite un suivi des priorités et une gestion des messages plus complexe.

6.3.2 Recommandations d'Utilisation

— **Algorithme de Chang et Roberts :** Il est bien adapté aux systèmes avec une topologie en anneau où les processus peuvent facilement déterminer leur successeur. Ce choix est idéal pour des systèmes avec peu de processus.
— **Algorithme de Franklin :** Ce dernier convient mieux aux environnements où une coordination stricte est nécessaire, avec une tolérance modérée aux pannes.
— **Algorithme de Bully :** Il est plus adapté aux systèmes où la priorisation entre les processus est cruciale, bien que son utilisation soit moins favorable dans les systèmes à grande échelle en raison de sa complexité.

En conclusion, le choix de l'algorithme dépend fortement du contexte applicatif. Pour des environnements avec peu de nœuds, Chang et Roberts sont préférés pour leur simplicité. Franklin se distingue par sa coordination méthodique, tandis que Bully excelle dans les environnements nécessitant une gestion stricte des priorités malgré une complexité accrue.

Quand choisir quel algorithme ?

— **Algorithme du Ring :** Cet algorithme est adapté pour des systèmes avec une structure circulaire prédéfinie et une faible tolérance aux pannes. Il convient aux systèmes où les pannes sont rares et où la simplicité de mise en œuvre est primordiale.
— **Algorithme de Bully :** Cet algorithme est préférable dans des systèmes où les processus sont identifiables par des numéros uniques et où une meilleure tolérance aux pannes est nécessaire. Cependant, il peut devenir coûteux en termes de messages échangés dans des systèmes de grande taille.

6.3.3 Critères d'évaluation

Les critères d'évaluation des algorithmes d'élection dans les systèmes distribués comprennent la latence, c'est-à-dire le temps nécessaire pour qu'un nœud devienne leader, ainsi que le nombre de messages échangés pendant le processus d'élection. La robustesse de l'algorithme face à des pannes ou à des comportements non attendus est également un critère clé. La simplicité de l'algorithme et sa capacité à s'adapter à des environnements distribués dynamiques sont également pris en considération. Enfin, la scalabilité, la jus-

tesse et l'équité de l'algorithme sont des critères importants à évaluer pour garantir un fonctionnement optimal dans différents contextes.

6.4 Applications et cas d'utilisation

Les algorithmes d'élection dans les systèmes distribués trouvent des applications diverses dans de nombreux domaines, tels que la gestion de réseaux de télécommunications, la supervision des systèmes informatiques en temps réel, la coordination des véhicules autonomes, et la synchronisation des processus dans les systèmes industriels. Leur capacité à élire un coordonnateur fiable et efficace au sein d'un réseau distribué en fait un outil précieux pour garantir une communication fluide et une prise de décision rapide.

6.4.1 Domaines industriels et scientifiques

Dans le domaine industriel, les algorithmes d'élection sont largement utilisés pour la coordination et la synchronisation des processus de fabrication automatisés, la gestion des systèmes de distribution d'énergie et la surveillance des réseaux de capteurs. Du point de vue scientifique, ces algorithmes sont essentiels pour la gestion des calculs distribués, la simulation des phénomènes naturels complexes, et la coordination des expériences dans des environnements de recherche collaboratifs.

6.5 Conclusion et perspectives

En conclusion, les algorithmes d'élection jouent un rôle essentiel dans les systèmes distribués en garantissant la désignation d'un coordinateur fiable et cohérent. Leur étude approfondie permet de mieux comprendre les défis et les enjeux liés à la gestion de ces systèmes. Pour l'avenir, il est crucial de continuer à explorer de nouvelles avancées dans le domaine des systèmes distribués afin de développer des algorithmes d'élection encore plus efficaces et robustes, capables de répondre aux besoins croissants des applications distribuées dans divers secteurs.

6.5.1 Avancées futures dans le domaine des systèmes distribués

Les avancées futures dans le domaine des systèmes distribués devraient se concentrer sur l'amélioration des performances et de la fiabilité des algorithmes d'élection. Il est nécessaire de poursuivre la recherche pour développer des stratégies plus sophistiquées et adaptatives, capables de gérer des environnements distribués complexes. De plus, l'intégration de concepts tels que l'apprentissage automatique et l'intelligence artificielle pourrait ouvrir de nouvelles perspectives pour la conception d'algorithmes d'élection innovants et adaptés aux besoins évolutifs des applications distribuées.

Chapitre 7

La mémoire virtuelle répartie et linéarisabilité

7.1 Introduction

La mémoire virtuelle répartie et la linéarisabilité sont des concepts fondamentaux dans les systèmes distribués modernes. Ce travail vise à explorer en profondeur ces deux concepts cruciaux pour la gestion efficace des données réparties. En comprenant les défis et les avantages de la mémoire virtuelle répartie ainsi que les différentes formes de linéarisabilité, il est possible de concevoir des systèmes plus robustes et fiables. Cette introduction donnera un aperçu des thèmes abordés dans ce document, en soulignant leur pertinence et leur impact sur les applications distribuées.

7.1.1 Définitions de base

Pour bien comprendre les concepts de mémoire virtuelle répartie et de linéarisabilité, il est essentiel de maîtriser certaines définitions de base. Cela inclut une compréhension claire de ce qu'est la mémoire virtuelle répartie, comment elle fonctionne, et les implications de son utilisation dans un environnement distribué. De même, la linéarisabilité nécessite une définition précise pour comprendre ses différentes formes et son importance dans la cohérence des données. Ce chapitre établira les bases nécessaires pour une analyse approfondie des sujets à venir.

7.2 Mémoire Virtuelle Répartie

La mémoire virtuelle répartie se réfère à la distribution des éléments de la mémoire sur différents nœuds d'un système distribué. Cela permet un accès rapide aux données et une meilleure utilisation des ressources. Cette approche nécessite une communication efficace entre les nœuds pour assurer la cohérence des données.

7.2.1 Concepts fondamentaux

Les concepts fondamentaux de la mémoire virtuelle répartie incluent la distribution transparente des données, la gestion de la cohérence et la tolérance aux pannes. Cela implique également la partition des données et l'accès concurrent aux ressources partagées, ce qui nécessite une coordination efficace entre les différents nœuds du système.

7.2.2 Architectures et modèles

Les architectures et modèles de la mémoire virtuelle répartie comprennent des approches telles que la réplication des données, la stratégie de cohérence et la gestion des verrous. Ces modèles visent à garantir l'intégrité des données tout en optimisant les performances du système distribué, en tenant compte des contraintes de communication et de latence.

7.3 Linéarisabilité dans les Systèmes Répartis

La linéarisabilité dans les systèmes répartis est un concept fondamental qui garantit l'ordre des opérations dans un système distribué. Cela implique que chaque opération s'effectue comme si elle était exécutée à un moment précis, respectant ainsi un ordre global. Pour assurer la linéarisabilité, il est essentiel de comprendre les fondements théoriques et les mécanismes sous-jacents à ce concept, ce qui sera exploré plus en détail dans les sections suivantes.

7.3.1 Fondements théoriques

Les fondements théoriques de la linéarisabilité dans les systèmes répartis reposent sur la garantie de l'ordre des opérations malgré la distribution des données. Cela implique d'étudier en profondeur les concepts de cohérence et de consensus pour assurer que toutes les opérations respectent un ordre global, même dans un environnement distribué. Comprendre ces fondements théoriques est crucial pour mettre en œuvre efficacement la linéarisabilité dans les systèmes répartis.

7.3.2 Linéarisabilité par Exclusion Mutuelle

La linéarisabilité par exclusion mutuelle vise à garantir un ordre global pour les opérations en limitant l'accès concurrentiel aux ressources partagées. En utilisant des mécanismes tels que les verrous ou les sémaphores, cette approche assure que les opérations s'exécutent de manière séquentielle, respectant ainsi la linéarisabilité. Comprendre comment appliquer ce concept dans un système réparti est essentiel pour maintenir la cohérence et l'ordre des opérations. L'algorithme suivant illustre une implémentation de l'exclusion mutuelle pour assurer la linéarisabilité dans un système distribué :

Algorithm 12 Exclusion mutuelle par verrouillage

1: **Entrée :** Un ensemble de processus $P_1, P_2, ..., P_n$ partageant une ressource commune.
2: **Sortie :** L'accès séquentiel à la ressource partagée en respectant la linéarisabilité.
3: **procedure** ACCÉDER À LA RESSOURCE
4: **demander un verrou** ▷ Vérification de l'accès exclusif à la ressource
5: **entrer dans la section critique** ▷ Accès à la ressource partagée
6: **exécuter l'opération** ▷ Effectuer l'opération souhaitée sur la ressource partagée
7: **libérer le verrou** ▷ Libération de la ressource pour les autres processus
8: **end procedure**

7.3.3 Linéarisabilité avec Gestionnaire Statique

La linéarisabilité avec un gestionnaire statique implique l'utilisation d'un ensemble prédéfini de ressources et de règles pour garantir un ordre linéaire pour les opérations dans un système réparti. En définissant clairement les règles et les priorités d'accès aux ressources, il est possible de maintenir la linéarisabilité tout en assurant une gestion statique des opérations. Comprendre comment configurer et utiliser un tel gestionnaire est crucial pour maintenir la cohérence des opérations dans un environnement distribué. L'algorithme suivant illustre l'utilisation d'un gestionnaire statique pour assurer la linéarisabilité des opérations dans un système distribué. Un gestionnaire statique de ressources garantit que les ressources sont utilisées selon des règles prédéfinies et un ordre d'accès déterminé.

Algorithm 13 Linéarisabilité avec Gestionnaire Statique

Entrée : Un ensemble de ressources $R_1, R_2, \ldots, R_m$ et un ensemble de processus $P_1, P_2, \ldots, P_n$.
Sortie : Les ressources sont utilisées de manière linéaire, respectant un ordre d'accès préétabli.
procedure GÉRER_RESSOURCES
    for chaque processus P_i dans $P_1, P_2, \ldots, P_n$ **do**
        vérifier la disponibilité de la ressource R_k ▷ Vérification de l'état de la ressource
        if la ressource R_k est disponible **then**
            allouer la ressource R_k à P_i
            effectuer l'opération sur la ressource R_k ▷ Le processus P_i réalise une opération sur la ressource
            libérer la ressource R_k ▷ Libération de la ressource après l'opération
        else
            attendre ▷ Le processus P_i attend que la ressource soit libérée
        end if
    end for
end procedure

7.3.4 Linéarisabilité avec Gestionnaire Dynamique

La linéarisabilité avec un gestionnaire dynamique implique la gestion flexible des ressources et des opérations tout en assurant un ordre global pour les opérations dans un système réparti. En permettant des ajustements dynamiques en fonction des besoins et des priorités, ce type de gestionnaire offre une approche adaptable pour maintenir la linéarisabilité tout en gérant efficacement les opérations dans un environnement distribué complexe. Comprendre comment mettre en œuvre et adapter un tel gestionnaire est essentiel pour assurer la cohérence et l'ordre des opérations. L'algorithme suivant illustre l'utilisation d'un gestionnaire dynamique pour assurer la linéarisabilité des opérations dans un système distribué tout en permettant des ajustements en fonction des besoins et des priorités.

Algorithm 14 Linéarisabilité avec Gestionnaire Dynamique

Entrée : Un ensemble de ressources $R_1, R_2, \ldots, R_m$ et un ensemble de processus $P_1, P_2, \ldots, P_n$, avec des priorités dynamiques.
Sortie : Les ressources sont utilisées de manière séquentielle, respectant un ordre d'accès global tout en permettant des ajustements dynamiques.
procedure GÉRER_RESSOURCES_DYNAMIQUE
    for chaque processus P_i dans $P_1, P_2, \ldots, P_n$ **do**
        dynamiser les priorités ▷ Ajustement dynamique des priorités d'accès
        évaluer la disponibilité des ressources R_k ▷ Vérification de l'état actuel de la ressource en fonction des priorités
        if la ressource R_k est disponible **then**
            allouer la ressource R_k à P_i
            effectuer l'opération sur la ressource R_k
            mettre à jour les priorités ▷ Mise à jour dynamique des priorités après l'opération
            libérer la ressource R_k
        else
            mettre en file d'attente P_i ▷ Si la ressource est occupée, le processus attend selon ses priorités
        end if
    end for
end procedure

7.4 Propagation des Écritures

La propagation des écritures dans les systèmes répartis est un élément crucial pour assurer la cohérence et la fiabilité des données. Ce processus permet de diffuser les mises à jour d'une donnée à tous les nœuds du système de manière synchronisée et ordonnée. La propagation des écritures peut se faire de manière synchrone ou asynchrone, en fonction des besoins de l'application et des contraintes de performance. Elle est également étroitement liée à la gestion de la cohérence des données et des mécanismes de contrôle de la concurrence. La propagation des écritures dans les systèmes répartis est un élément crucial pour assurer la cohérence et la fiabilité des données. Ce processus permet de diffuser les mises à jour d'une donnée à tous les nœuds du système de manière synchronisée et ordonnée. La propagation des écritures peut se faire de manière synchrone ou asynchrone, en fonction des besoins de l'application et des contraintes de performance. Elle est également étroitement liée à la gestion de la cohérence des données et des mécanismes de contrôle de la concurrence.

7.4.1 Mécanismes de Propagation

Les mécanismes de propagation des écritures comprennent différentes stratégies pour diffuser les mises à jour à travers le système réparti. Parmi ces mécanismes, on retrouve la diffusion en mode push, où les mises à jour sont directement transmises aux nœuds voisins, et la diffusion en mode pull, où les nœuds demandent activement les mises à jour aux autres nœuds. Il existe également des mécanismes hybrides qui combinent ces deux approches pour optimiser la propagation des écritures tout en garantissant la cohérence

Algorithm 15 Propagation des Écritures

Entrée : Une donnée D à mettre à jour, un ensemble de nœuds $N_1, N_2, \ldots, N_m$ dans le système.
Sortie : La donnée D est mise à jour sur tous les nœuds du système.
procedure PROPAGER_ECRITURE
    for chaque nœud N_i dans $N_1, N_2, \ldots, N_m$ **do**
        if mode de propagation = synchrone **then**
            mettre à jour la donnée D sur le nœud N_i
            attendre la confirmation de N_i avant de passer au nœud suivant
        else if mode de propagation = asynchrone **then**
            mettre à jour la donnée D sur le nœud N_i
            continuer avec le prochain nœud sans attendre de confirmation
        end if
    end for
    if mode de propagation = synchrone **then**
        attendre les confirmations de tous les nœuds avant de considérer l'écriture comme terminée
    end if
end procedure

des données.

7.4.2 Consistance des Données

La consistance des données dans le contexte de la propagation des écritures fait référence à la garantie que les mises à jour sont appliquées de manière cohérente et dans le bon ordre à tous les nœuds du système. Cette cohérence peut être assurée par des protocoles de réplication de données, des mécanismes de verrouillage ou des algorithmes de contrôle de la concurrence. La garantie de la consistance des données est essentielle pour éviter les incohérences et les conflits dans les systèmes répartis, en particulier lorsque plusieurs nœuds tentent de modifier la même donnée simultanément.

7.5 Conclusion et Perspectives

En conclusion, ce travail a permis de mettre en évidence l'importance de la linéarisabilité dans les systèmes répartis, en mettant en lumière les différentes approches telles que l'exclusion mutuelle, le gestionnaire statique et le gestionnaire dynamique. De plus, nous avons exploré les mécanismes de propagation des écritures ainsi que la consistance des données, soulignant l'impact de ces éléments sur la mémoire virtuelle répartie. Pour l'avenir, il serait bénéfique de poursuivre les recherches dans ce domaine afin de développer des modèles plus efficaces pour la gestion de la mémoire virtuelle répartie, ainsi que pour affiner les techniques de linéarisabilité dans les systèmes répartis.

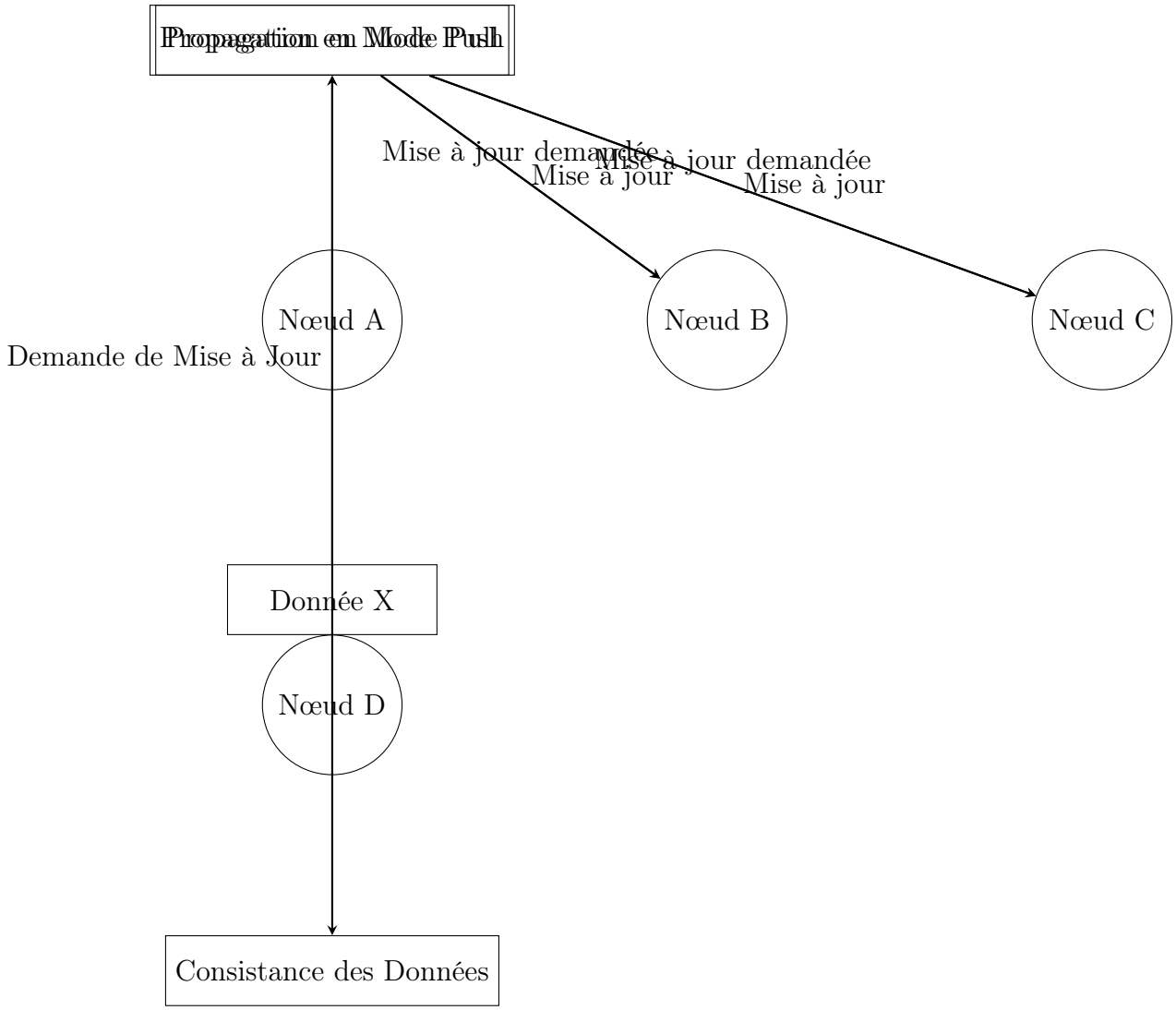


FIGURE 7.1 – Mécanismes de Propagation dans les Systèmes Distribués

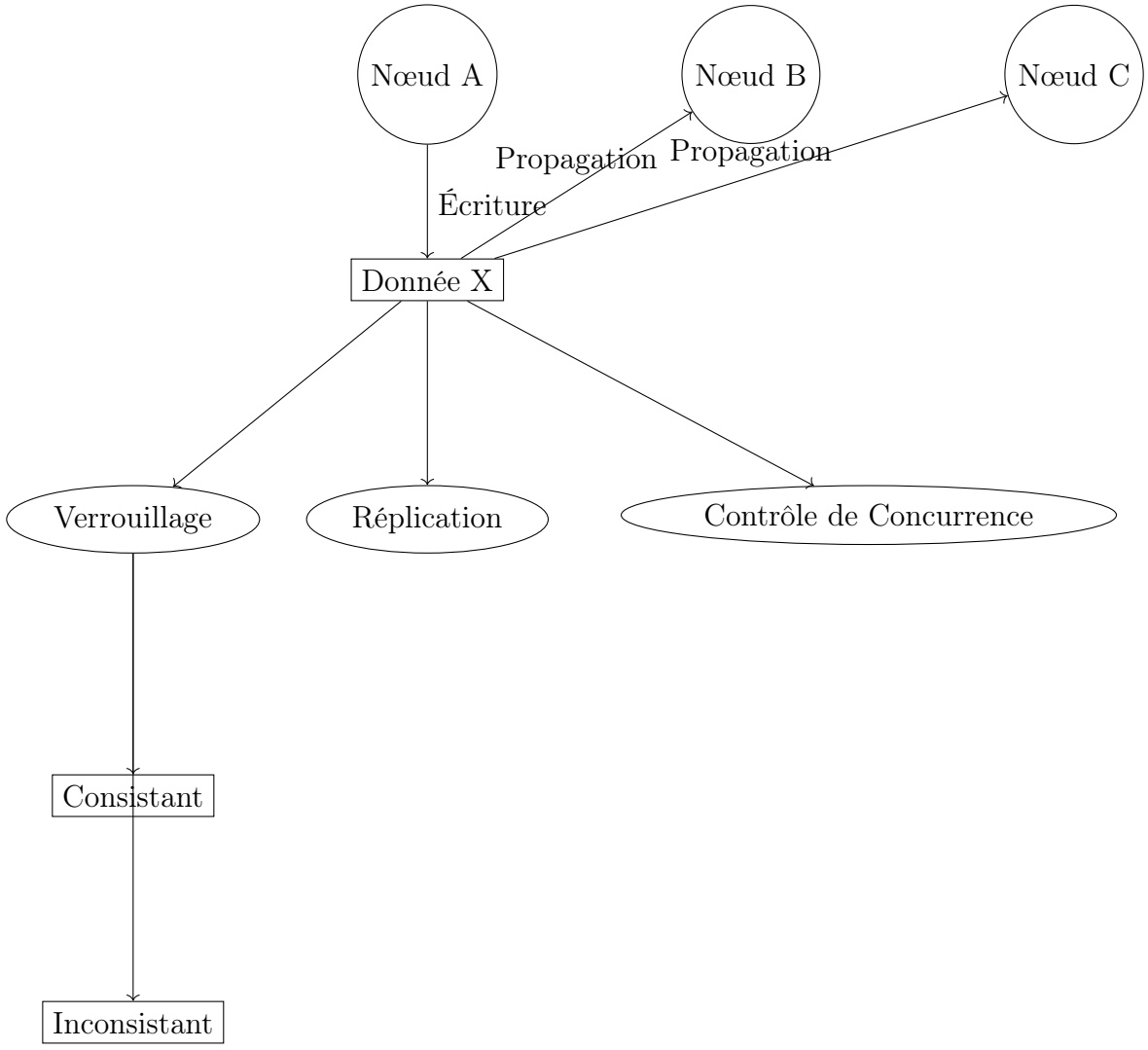


FIGURE 7.2 – Consistance des Données dans les Systèmes Distribués

Chapitre 8

L'autostabilisation

8.1 Introduction à l'autostabilisation

L'introduction à l'autostabilisation met en lumière la capacité des systèmes répartis à atteindre un état stable malgré des conditions initiales arbitraires. Ce concept est essentiel pour assurer la fiabilité et la résilience des systèmes distribués, notamment dans des environnements sujets aux pannes et aux perturbations. En comprenant les principes fondamentaux de l'autostabilisation, il devient possible de concevoir des systèmes capables de se rétablir automatiquement, garantissant ainsi un fonctionnement continu et robuste.

8.1.1 Concepts de base

Les concepts de base de l'autostabilisation incluent la notion d'état global cohérent, la convergence vers un état stable, ainsi que la résilience aux fautes et aux perturbations. Comprendre ces concepts est crucial pour développer des algorithmes et des protocoles capables de garantir l'autostabilisation des systèmes répartis. En se basant sur ces fondements, il devient possible de mettre en place des mécanismes de routage auto-stabilisants, de gérer la mémoire virtuelle de manière distribuée et d'assurer l'exclusion mutuelle sans nécessiter de supervision externe.

8.1.2 Historique et évolution

L'historique de l'autostabilisation dans les systèmes répartis remonte aux premières études sur les réseaux de communication et les systèmes informatiques distribués. Au fil du temps, l'autostabilisation a évolué pour devenir un concept essentiel dans la conception et la gestion des systèmes distribués. Depuis ses premières applications théoriques jusqu'à ses implémentations pratiques, l'autostabilisation a connu un développement significatif, devenant un pilier fondamental des technologies de réseaux et de communication modernes.

8.2 Fondements théoriques de l'autostabilisation

La section des fondements théoriques de l'autostabilisation explore les principes fondamentaux de ce concept dans les systèmes répartis. Elle examine les modèles de systèmes répartis, mettant en lumière les différentes structures et configurations qui influent sur l'autostabilisation. De plus, cette section aborde également les automates finis, qui sont

des outils cruciaux pour la compréhension et l'implémentation de l'autostabilisation dans les systèmes répartis.

8.2.1 Modèles de systèmes répartis

La sous-section sur les modèles de systèmes répartis se concentre sur les différents modèles théoriques qui servent de fondement à l'autostabilisation. Elle passe en revue les modèles de communication, de synchronisation et de distribution des ressources, soulignant leur impact sur la capacité des systèmes répartis à s'autostabiliser. En outre, elle explore les caractéristiques clés de ces modèles qui favorisent ou entravent l'autostabilisation.

8.2.2 Automates finis

La sous-section sur les automates finis examine en détail ces dispositifs de calcul finis, leurs caractéristiques et leur application dans l'autostabilisation des systèmes répartis. Elle présente les concepts de base des automates finis et leur utilisation pour modéliser les comportements des systèmes répartis lors du processus d'autostabilisation. De plus, elle met en évidence les différentes classes d'automates finis et leur pertinence pour l'autostabilisation.

8.3 Routage auto-stabilisant

Dans le contexte des systèmes répartis, le routage auto-stabilisant vise à permettre aux données de trouver leur chemin de manière autonome, même en cas de perturbations ou de dysfonctionnements. Ce concept repose sur l'idée que le système doit être capable de se rétablir de manière automatique, sans intervention extérieure, pour maintenir la connectivité et la communication. Il s'agit donc d'un élément essentiel pour assurer la fiabilité et la résilience des systèmes distribués.

8.3.1 Principes de base

Les principes de base du routage auto-stabilisant incluent la décentralisation, la redondance et l'auto-correction. En d'autres termes, chaque nœud du réseau doit être en mesure de prendre des décisions indépendamment, le réseau doit être capable de tolérer la perte ou l'ajout de nœuds de manière transparente, et les erreurs ou les incohérences doivent être automatiquement corrigées sans perturber le fonctionnement global. Ces principes visent à garantir la résilience et la stabilité du système dans des conditions changeantes.

8.4 Algorithmes de Routage Auto-Stabilisants

Les **algorithmes de routage auto-stabilisants** sont conçus pour mettre en œuvre les principes de base du routage auto-stabilisant. Ils utilisent des mécanismes de détection d'anomalies, de recalibrage dynamique des chemins de communication et de gestion de la redondance pour assurer un fonctionnement continu et fiable du réseau. Ces algorithmes reposent sur des modèles mathématiques et des automates finis pour garantir des performances cohérentes et prévisibles, même en cas de variations ou de perturbations.

Principe de Fonctionnement des Algorithmes de Routage Auto-Stabilisants

1. **Détection d'anomalies** : Un mécanisme de détection d'anomalies est mis en place pour surveiller les comportements anormaux dans le réseau, comme les pannes de nœuds, les chemins de communication invalides ou les congestions. Ce mécanisme utilise des informations locales, généralement en surveillant les voisins directs ou les métriques de performance.

2. **Recalibrage dynamique des chemins** : Lorsqu'une anomalie est détectée (par exemple, un chemin de routage devient invalide ou un nœud est défaillant), l'algorithme ajuste automatiquement les chemins de communication pour rétablir une route valide. Cela peut inclure la redirection des paquets vers des chemins alternatifs ou la mise à jour des tables de routage locales pour intégrer les nouveaux chemins disponibles.

3. **Gestion de la redondance** : Les algorithmes de routage auto-stabilisants prennent en compte la redondance dans le réseau pour éviter la surcharge sur un chemin unique. Ils utilisent des routes multiples et choisissent dynamiquement la meilleure route en fonction des conditions du réseau.

4. **Stabilisation après perturbation** : Une fois que l'algorithme détecte une anomalie et ajuste les chemins de routage, il s'assure que les modifications apportées permettent au réseau de retrouver un état stable. Cela peut inclure l'échange de messages entre les nœuds pour partager les nouvelles informations de routage.

5. **Utilisation d'automates finis et de modèles mathématiques** : Les algorithmes de routage auto-stabilisants reposent souvent sur des modèles mathématiques comme des automates finis pour garantir que le processus de routage converge rapidement vers un état stable et prévisible, même après une perturbation.

Exemple d'Algorithme de Routage Auto-Stabilisant

Algorithm 16 Algorithme de Routage Auto-Stabilisant

Entrée : Un réseau de nœuds avec des chemins de communication, des tables de routage locales et des informations de voisinage.
Sortie : Un chemin de routage validé et optimisé, même en présence d'anomalies.
procedure ROUTAGE_AUTOSTABILISANT
    Initialiser les tables de routage locales avec les chemins disponibles.
    for chaque nœud N_i **do**
        Détecter anomalies {pannes, congestion, chemins invalides} ▷ Vérification de l'état des voisins et des chemins
        if une anomalie est détectée **then**
            Ajuster les chemins {mettre à jour les routes vers les nœuds voisins valides}
            Mise à jour de la table de routage {ajuster les distances, ajouter des chemins alternatifs}
        else
            Maintenir le chemin optimal {pas de changement nécessaire}
        end if
    end for
    Répéter le processus de stabilisation jusqu'à ce que les chemins convergent.
end procedure

Explication de l'Algorithme :

1. **Initialisation des tables de routage locales** : Chaque nœud initialise sa table de routage avec les chemins de communication valides disponibles au départ. Cela inclut la configuration initiale des voisins et des chemins accessibles.

2. **Détection des anomalies** : À chaque itération, chaque nœud vérifie l'état de ses voisins pour détecter toute anomalie. Cela peut inclure des pannes de nœuds, des chemins de routage invalides ou des congestions de réseau. Les informations de voisinage sont utilisées pour cette détection.

3. **Ajustement dynamique des chemins** : Lorsqu'une anomalie est détectée, le nœud ajuste ses chemins de communication. Par exemple, il peut ajouter un chemin alternatif ou recalibrer les distances vers ses voisins pour assurer un routage fiable. Les tables de routage sont mises à jour pour intégrer ces changements.

4. **Maintien des chemins optimaux** : Si aucune anomalie n'est détectée, le nœud maintient les chemins de routage existants. Cela permet au réseau de fonctionner de manière stable lorsqu'il n'y a pas de perturbations.

5. **Répétition et stabilisation** : Le processus est répété à intervalles réguliers jusqu'à ce que le système converge vers une configuration stable. Cela permet au réseau de se réadapter en temps réel aux conditions de communication changeantes.

Garantie de Stabilité et Cohérence

L'algorithme garantit la stabilité et la cohérence du réseau en utilisant des mises à jour dynamiques des chemins. Chaque nœud ajuste ses routes en réponse aux perturbations, et le processus de stabilisation permet de rétablir un fonctionnement normal après chaque perturbation. L'utilisation d'automates finis et de modèles mathématiques aide à assurer que les modifications apportées aux chemins convergent rapidement et que le réseau reste cohérent malgré les variations.

Applications et Avantages :

- **Réseaux ad-hoc et sans fil** : Dans des environnements où la topologie du réseau change fréquemment, les algorithmes de routage auto-stabilisants sont cruciaux pour maintenir une connectivité fiable. - **Réseaux de capteurs** : Dans des réseaux de capteurs distribués, ces algorithmes permettent de maintenir un routage efficace même en cas de défaillances de capteurs ou de perturbations. - **Réseaux de communication** : Ils assurent la résilience du réseau en permettant une réponse dynamique aux défaillances des chemins et des nœuds.

En résumé, les **algorithmes de routage auto-stabilisants** sont essentiels pour maintenir une performance fiable et stable dans les systèmes distribués, en particulier dans des réseaux où les conditions changent fréquemment ou des perturbations se produisent. Ces algorithmes permettent d'assurer une gestion dynamique des chemins de routage tout en garantissant la cohérence et la fiabilité du réseau.

8.5 Gestion de la mémoire virtuelle répartie

La gestion de la mémoire virtuelle répartie est essentielle dans les systèmes répartis pour permettre l'accès à des données partagées de manière efficace et fiable. Elle joue un rôle crucial dans la résolution des problèmes liés à la cohérence des données et à la synchronisation entre les différents nœuds du système. En assurant une allocation et une libération optimales de la mémoire, la gestion de la mémoire virtuelle répartie contribue à améliorer les performances et la stabilité du système dans son ensemble.

8.5.1 Nécessité et enjeux

La nécessité de la gestion de la mémoire virtuelle répartie réside dans le fait que les systèmes répartis doivent gérer des ressources partagées de manière transparente pour les utilisateurs. Les enjeux sont nombreux, allant de la garantie de l'intégrité des données à la minimisation des conflits d'accès, en passant par la limitation de l'impact sur les performances du système. La nécessité de répondre à ces enjeux est primordiale pour assurer un fonctionnement cohérent et fiable du système distribué.

8.5.2 Techniques et protocoles

Les techniques et protocoles de gestion de la mémoire virtuelle répartie incluent la mise en œuvre de mécanismes de partage et de protection des ressources, tels que la segmentation, la pagination et la migration de pages. Des protocoles de cohérence de la mémoire et des mécanismes de communication entre les nœuds sont également essentiels pour assurer l'intégrité des données partagées. Ces techniques et protocoles visent à garantir un accès efficace et sécurisé à la mémoire virtuelle répartie dans les systèmes distribués.

8.6 Exclusion mutuelle

L'exclusion mutuelle dans les systèmes répartis est un concept fondamental qui vise à garantir qu'à tout moment, un seul processus puisse accéder à une ressource partagée. Cela évite les conflits et assure la cohérence des données. La mise en place de mécanismes d'exclusion mutuelle auto-stabilisants est essentielle pour assurer le bon fonctionnement des systèmes répartis, même en cas de perturbations ou de dysfonctionnements.

8.6.1 Problématique de l'exclusion mutuelle

La problématique de l'exclusion mutuelle réside dans la nécessité de garantir l'accès exclusif à une ressource partagée, tout en tenant compte des éventuelles défaillances et des perturbations. Les systèmes répartis doivent être capables de maintenir cette propriété malgré les conditions instables ou les pannes temporaires. Cela nécessite une approche spécifique pour concevoir des algorithmes auto-stabilisants efficaces.

8.7 Algorithmes d'Exclusion Mutuelle Auto-Stabilisants

Les **algorithmes d'exclusion mutuelle auto-stabilisants** sont conçus pour garantir que chaque processus respecte les règles d'exclusion mutuelle, même en présence de fautes ou d'incohérences. Ces algorithmes doivent être capables de détecter et de corriger les violations de l'exclusion mutuelle de manière autonome, sans nécessiter d'intervention extérieure. Cela permet d'assurer la robustesse et la fiabilité des systèmes répartis.

Principe de Fonctionnement des Algorithmes d'Exclusion Mutuelle Auto-Stabilisants

1. **Exclusion mutuelle** : L'exclusion mutuelle garantit qu'à tout moment, un seul processus peut accéder à une ressource partagée. Cela évite les conflits et garantit la cohérence des opérations.

2. **Détection et correction des violations** : En cas de violation de l'exclusion mutuelle, l'algorithme doit être capable de détecter la violation et de rétablir l'ordre sans

intervention externe. Les violations peuvent inclure des processus accédant simultanément à une ressource ou des erreurs de synchronisation.

3. **Auto-stabilisation** : Ces algorithmes doivent converger rapidement vers un état stable où les processus respectent les règles d'exclusion mutuelle. Une fois l'anomalie corrigée, le système doit revenir automatiquement à un état stable sans nécessiter d'action supplémentaire.

4. **Robustesse** : L'algorithme doit être capable de fonctionner de manière fiable malgré des défaillances de nœuds, des pertes de messages ou d'autres perturbations du réseau.

5. **Utilisation de marqueurs de contrôle et de protocoles de synchronisation** : Ces algorithmes utilisent des mécanismes tels que des marqueurs de contrôle (qui suivent l'état des ressources partagées) et des protocoles de synchronisation pour garantir que les processus respectent les règles d'exclusion mutuelle.

Exemple d'Algorithme d'Exclusion Mutuelle Auto-Stabilisant

L'algorithme suivant illustre un modèle simplifié d'un algorithme d'exclusion mutuelle auto-stabilisant, utilisant des sémaphores pour garantir que l'accès à une ressource partagée est exclusif.

Algorithm 17 Algorithme d'Exclusion Mutuelle Auto-Stabilisant

Entrée : Un réseau de processus $P_1, P_2, \ldots, P_n$ partageant une ressource commune, avec des sémaphores pour gérer l'accès exclusif.
Sortie : Un accès séquentiel à la ressource, avec détection et correction des violations d'exclusion mutuelle.
procedure EXCLUSION_MUTUELLE_AUTOSTABILISANT
    for chaque processus P_i dans $P_1, P_2, \ldots, P_n$ **do**
        demander un sémaphore pour accéder à la ressource ▷ L'attente garantit l'exclusion mutuelle
        entrer dans la section critique ▷ Accéder à la ressource partagée
        effectuer l'opération sur la ressource ▷ Opération effectuée en section critique
        libérer le sémaphore ▷ Libérer l'accès à la ressource pour les autres processus
        if une violation d'exclusion mutuelle est détectée **then**
            réinitialiser les sémaphores ▷ Réparation de l'anomalie et rétablissement de l'exclusion mutuelle
        end if
    end for
    Répéter le processus jusqu'à ce que l'état stable soit atteint.
end procedure

Explication de l'Algorithme :

1. **Demander un sémaphore** : Chaque processus P_i demande un sémaphore pour accéder à la ressource partagée. Si un autre processus détient déjà le sémaphore, le processus attend. Cela garantit qu'un seul processus peut accéder à la ressource à un instant donné, respectant ainsi l'exclusion mutuelle.

2. **Entrée dans la section critique** : Une fois que le processus a obtenu le sémaphore, il entre dans la section critique et accède à la ressource partagée. Cela garantit qu'aucun autre processus ne peut accéder à la ressource en même temps.

3. **Effectuer l'opération sur la ressource** : Le processus effectue l'opération souhaitée sur la ressource partagée.

4. **Libération du sémaphore** : Après avoir terminé l'opération, le processus libère le sémaphore, permettant à d'autres processus d'y accéder.

5. **Détection des violations et réinitialisation des sémaphores** : Si une violation de l'exclusion mutuelle est détectée, l'algorithme réinitialise les sémaphores et rétablit un état stable. Cela permet de corriger les anomalies de manière autonome.

6. **Stabilisation** : Le processus est répété de manière à ce que l'algorithme converge vers un état stable, garantissant le respect de l'exclusion mutuelle malgré les perturbations ou défaillances du réseau.

Garantie de Stabilité et Cohérence

L'algorithme assure la stabilité et la cohérence du système en permettant une gestion autonome des violations d'exclusion mutuelle. Lorsqu'une anomalie est détectée, elle est corrigée et l'algorithme se stabilise automatiquement. Cela permet aux systèmes distribués d'être plus robustes et de fonctionner de manière fiable, même en présence de fautes ou de perturbations.

Applications et Avantages :

- **Systèmes répartis et réseaux** : Ces algorithmes sont utilisés pour garantir que plusieurs processus ne tentent pas d'accéder simultanément à une ressource partagée, ce qui est crucial dans les systèmes de gestion de bases de données ou de fichiers distribués. - **Systèmes tolérants aux fautes** : Ces algorithmes garantissent que le système continue de fonctionner correctement même en cas de défaillance de certains processus ou nœuds. - **Amélioration de la performance et de la fiabilité** : En assurant l'exclusion mutuelle tout en étant capables de se stabiliser en cas de violations, ces algorithmes améliorent la performance et la fiabilité globale du système.

Les **algorithmes d'exclusion mutuelle auto-stabilisants** sont essentiels dans les systèmes distribués pour garantir la cohérence des ressources partagées. Grâce à leur capacité à détecter et corriger les violations de l'exclusion mutuelle, ils assurent une stabilité et une performance continues, même face aux défaillances et perturbations. Ces algorithmes sont indispensables pour la gestion robuste des ressources dans les réseaux distribués.

8.8 Applications de l'autostabilisation

Les applications de l'autostabilisation sont vastes et variées, touchant de nombreux domaines, tels que les réseaux de capteurs, les systèmes de communication, la gestion de la mémoire virtuelle répartie, et le routage auto-stabilisant. Ces applications offrent des solutions robustes et fiables pour assurer le bon fonctionnement des systèmes répartis, même en cas de perturbations ou de changements dynamiques. L'autostabilisation joue un rôle essentiel dans la conception et le déploiement de ces applications, offrant une garantie de correction sans intervention extérieure.

8.8.1 Réseaux de capteurs

Les réseaux de capteurs représentent l'un des principaux domaines d'application de l'autostabilisation. Grâce à cette technique, il est possible d'assurer un fonctionnement fiable des capteurs, même dans des environnements hostiles ou instables. Les algorithmes auto-stabilisants permettent de détecter et de corriger automatiquement les éventuelles

défaillances des capteurs, garantissant ainsi une collecte de données précise et continue, essentielle pour de nombreuses applications telles que la surveillance environnementale, l'agriculture de précision, ou la gestion des infrastructures urbaines.

8.8.2 Systèmes de communication

Dans le domaine des systèmes de communication, l'autostabilisation offre des avantages significatifs en termes de fiabilité et de tolérance aux pannes. Les protocoles et les algorithmes auto-stabilisants permettent de maintenir la connectivité et la qualité de service dans des réseaux dynamiques et sujets à des changements imprévisibles. Ces systèmes peuvent ainsi s'adapter de manière autonome aux conditions changeantes, assurant une communication continue et efficace, même en présence de perturbations ou de défaillances.

8.9 Défis et perspectives de recherche

Les défis et perspectives de recherche dans le domaine de l'autostabilisation des systèmes répartis sont multiples et variés. Il est essentiel de poursuivre les travaux pour améliorer la fiabilité, la performance et la résilience des systèmes auto-stabilisants. De plus, la prise en compte des contraintes liées à l'utilisation efficiente des ressources et à la sécurité constitue un enjeu majeur. Enfin, la recherche doit également se concentrer sur l'adaptation des systèmes auto-stabilisants à l'évolution des environnements, notamment avec l'émergence de nouvelles technologies et usages.

8.9.1 Limitations actuelles

Les limitations actuelles de l'autostabilisation dans les systèmes répartis résident principalement dans la complexité des algorithmes et des protocoles, qui peuvent rendre la mise en œuvre et la maintenance des systèmes auto-stabilisants difficiles. De plus, la gestion des erreurs et des défaillances reste un défi, en particulier dans des environnements dynamiques et hétérogènes. Enfin, la scalabilité des systèmes auto-stabilisants constitue également une limitation importante, surtout dans le contexte des réseaux de grande taille et des applications à grande échelle.

8.9.2 Axes de recherche futurs

Les axes de recherche futurs dans le domaine de l'autostabilisation des systèmes répartis devraient inclure des travaux visant à simplifier la conception et la mise en œuvre des systèmes auto-stabilisants, par le développement de modèles et d'outils adaptés. De plus, la recherche devrait se concentrer sur l'élaboration de mécanismes de détection et de récupération d'erreurs plus efficaces, permettant d'améliorer la robustesse des systèmes auto-stabilisants. Enfin, l'exploration de nouvelles approches pour assurer la scalabilité et l'adaptabilité des systèmes auto-stabilisants représente un axe crucial pour les recherches à venir.

Chapitre 9

Conclusion Générale

L'achèvement de ce module sur les Algorithmes et Systèmes Répartis marque une étape cruciale dans la formation des étudiants de Master 1 en Réseaux et Systèmes d'Information Distribués (RSID), Systèmes d'Information Décisionnels (SID), et Intelligence Artificielle et Analyse de Données (IAA). À travers l'exploration des principes fondamentaux des systèmes répartis et des algorithmes associés, les étudiants ont acquis non seulement des compétences techniques mais aussi une compréhension approfondie des enjeux contemporains liés à cette discipline.

Au cours de ce parcours académique, nous avons abordé une variété de sujets essentiels, allant des architectures de systèmes répartis aux algorithmes fondamentaux, en passant par les défis liés à la tolérance aux pannes et à la sécurité des systèmes. Cette diversité de contenu a permis aux étudiants de développer une vision globale des systèmes répartis, en les préparant à faire face aux défis complexes du monde professionnel.

L'un des aspects clés abordés dans ce module est la notion de résilience dans les systèmes répartis. La capacité d'un système à continuer à fonctionner de manière efficace malgré des pannes ou des défaillances est cruciale pour de nombreuses applications modernes. Les étudiants ont appris à concevoir des systèmes qui intègrent des mécanismes de tolérance aux pannes, garantissant ainsi la continuité des services et la fiabilité des données. Cela est d'autant plus pertinent à l'heure où les entreprises reposent de plus en plus sur des infrastructures distribuées pour leurs opérations quotidiennes.

En outre, la sécurité des systèmes répartis a été un sujet central de notre étude. Dans un contexte où les cybermenaces sont en constante évolution, il est essentiel que les futurs professionnels soient équipés de connaissances approfondies en matière de sécurité des données. Les discussions autour des meilleures pratiques en matière de sécurité ont permis aux étudiants de prendre conscience des enjeux liés à la protection des informations sensibles et de développer des compétences pratiques pour mettre en œuvre des solutions sécurisées.

L'intégration de l'intelligence artificielle et de l'apprentissage automatique dans le domaine des systèmes répartis constitue une avancée passionnante et innovante. Les étudiants ont eu l'occasion d'explorer comment ces technologies peuvent améliorer les performances des systèmes répartis, en facilitant des processus décisionnels plus efficaces et en permettant une meilleure prédiction des défaillances. Cette convergence entre l'IA et les systèmes répartis ouvre de nouvelles perspectives pour l'innovation technologique, et les étudiants sont ainsi préparés à contribuer activement à cette évolution.

Les projets pratiques et les études de cas réalisés tout au long du module ont également joué un rôle crucial dans l'apprentissage. En travaillant sur des problématiques réelles, les

étudiants ont pu appliquer les concepts théoriques à des situations concrètes, développant ainsi des compétences en résolution de problèmes et en travail d'équipe. Cette approche pratique renforce non seulement la compréhension des concepts mais prépare également les étudiants à s'adapter rapidement aux exigences du marché du travail.

À l'issue de ce module, les étudiants sont mieux préparés à relever les défis associés aux systèmes répartis dans des domaines variés tels que les télécommunications, l'Internet des objets (IoT), l'industrie 4.0, et bien d'autres. Ils disposent des outils nécessaires pour concevoir, analyser et optimiser des systèmes complexes, tout en prenant en compte les enjeux de performance, de sécurité, et de fiabilité. Cette formation leur permettra de devenir des acteurs clés dans le développement de solutions innovantes et efficaces dans un monde de plus en plus interconnecté.

En conclusion, le module "Algorithmes et Systèmes Répartis" représente un socle solide sur lequel les étudiants peuvent construire leur carrière professionnelle. Les connaissances acquises, combinées à une approche pratique et collaborative, les prépareront à naviguer avec succès dans un environnement technologique en constante évolution. En tant que futurs ingénieurs et chercheurs, ils auront la responsabilité de contribuer à l'avancement des systèmes répartis, tout en veillant à ce que ces technologies soient utilisées de manière éthique et responsable. Il est essentiel que les étudiants continuent à explorer et à se perfectionner dans ce domaine dynamique, afin de rester à la pointe de l'innovation et de répondre aux besoins croissants de notre société numérique.

Nous espérons que ce polycopié servira de ressource précieuse tout au long de leur parcours académique et professionnel. Que chaque étudiant trouve l'inspiration nécessaire pour poursuivre ses ambitions et contribuer de manière significative au monde des systèmes répartis.

Chapitre 10

Exercices sur la Communication

Exercice 1 : Contrôle de Flux

Énoncé : Dans un système de communication, le processus A envoie des messages à un processus B. Si A envoie un message alors que B est occupé à traiter un message précédent, que se passe-t-il ? Proposez une méthode simple de contrôle de flux pour éviter cette situation. Expliquez le mécanisme.

Corrigé : Lorsque A envoie un message à B alors que B est occupé, cela peut entraîner la perte de messages ou un débordement de la mémoire tampon de B. Une méthode simple de contrôle de flux consiste à utiliser des mécanismes d'accusé de réception (ACK).

Mécanisme proposé :

- **Accusé de réception :** A n'envoie un nouveau message que lorsqu'il reçoit un accusé de réception du message précédent envoyé à B.
- **Tampon :** B doit maintenir un tampon pour stocker les messages entrants jusqu'à ce qu'ils puissent être traités.
- **Fenêtre glissante :** Utiliser une approche de fenêtre glissante où A peut envoyer plusieurs messages, mais ne peut avancer la fenêtre que lorsque B a accusé réception de ces messages.

Exercice 2 : Communication Synchrone avec Rendez-vous

Énoncé : Considérez un modèle de communication synchrone avec rendez-vous entre deux processus, A et B. A doit envoyer une requête à B, et B doit répondre avant qu'A puisse continuer. Écrivez un pseudo-code pour illustrer ce processus.

—

Corrigé : Voici un exemple de pseudo-code :

[language=Python, caption=Pseudo-code de Communication Synchrone] Processus A : début envoyer demande à B attendre réponse de B traiter réponse fin

Processus B : début attendre demande de A traiter demande envoyer réponse à A fin

—

Explication : A envoie une demande à B et attend sa réponse avant de poursuivre. Cela garantit que les deux processus sont synchronisés.

Mesure du Temps de Traitement : Le temps total est la somme du temps de traitement de la requête par B et de la réponse par A. Ce modèle permet d'évaluer

précisément les temps de traitement individuels pour chaque étape de la communication, tout en assurant la coordination des processus.

Exercice 3 : Qualité de Service - Réseau FIFO

Énoncé : Décrivez ce qu'est un réseau FIFO (First In, First Out) et discutez des implications de ce type de réseau sur la qualité de service (QoS). Proposez une situation d'application.

Corrigé : Un réseau FIFO est un modèle de transmission où les messages sont traités dans l'ordre où ils arrivent. Cela signifie que le premier message envoyé est le premier à être reçu et traité.

Implications de QoS :

— **Latence :** La latence est prévisible, car l'ordre d'arrivée est garanti.

— **Bande passante :** La bande passante doit être suffisante pour traiter les messages dans un délai raisonnable.

— **Gigue :** La gigue est minimale car les messages sont traités dans un ordre séquentiel.

Situation d'application : Un exemple d'application pourrait être un système de messagerie instantanée où les messages doivent arriver dans l'ordre pour maintenir la cohérence de la conversation. Un réseau FIFO garantirait que les messages ne sont pas réarrangés, préservant ainsi le contexte de la communication.

Exercice 4 : Mise en Pratique des Concepts de Communication

Énoncé : Implémentez un protocole de communication simple entre deux processus, A et B, qui utilise le contrôle de flux et la communication synchrone avec rendez-vous. Rédigez le code en pseudo-langage.

Corrigé : [language=Python, caption=Pseudo-code de protocole de communication] Processus A : tampon = [] // Liste pour stocker les messages début pour chaque message dans messages à envoyer : ajouter message à tampon envoyer message à B attendre réponse de B retirer message de tampon fin

Processus B : début tant que vrai : attendre message de A traiter message envoyer accusé de réception à A fin

Explication : A utilise un tampon pour stocker les messages jusqu'à ce qu'il reçoive un accusé de réception de B. Cela permet de gérer le contrôle de flux. B traite les messages un par un et envoie un accusé de réception à A, ce qui assure la communication synchrone et le bon déroulement de l'échange.

Exercice 5 : Contrôle de Flux - Fenêtre Glissante

Énoncé :

Dans un protocole de communication utilisant une méthode de contrôle de flux par fenêtre glissante, la taille de la fenêtre est de 4. Étant donné les messages suivants : M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8 :

1. Indiquez l'ordre d'envoi des messages et le moment où chaque message est accusé de réception.
2. Si le message M_3 est perdu, que se passe-t-il ?

—

Corrigé :
1. **Ordre d'envoi des messages :**
— À l'instant $t = 0$: Envoi de M_1, M_2, M_3, M_4 (fenêtre pleine).
— À l'instant $t = 1$: Réception de l'accusé de réception pour M_1. Envoi de M_5.
— À l'instant $t = 2$: Réception de l'accusé de réception pour M_2. Envoi de M_6.
— À l'instant $t = 3$: Réception de l'accusé de réception pour M_3. Envoi de M_7.
— À l'instant $t = 4$: Réception de l'accusé de réception pour M_4. Envoi de M_8.
2. **Impact de la perte de M_3 :**
— La perte de M_3 signifie que le récepteur n'envoie pas l'accusé de réception pour ce message. Par conséquent, l'émetteur ne pourra pas avancer la fenêtre.
— Les messages M_5, M_6, M_7, et M_8 ne pourront pas être envoyés tant que M_3 n'aura pas été correctement reçu, **après sa retransmission suite à la détection d'un timeout**.
— Lorsqu'un timeout est détecté, l'émetteur réémet M_3 et redémarre le mécanisme de contrôle de flux. Une fois M_3 correctement accusé, la fenêtre peut avancer, permettant l'envoi des messages restants.

—

Conclusion : La méthode de contrôle de flux par fenêtre glissante assure la livraison ordonnée des messages et permet la retransmission des messages perdus. Le mécanisme de timeout est essentiel pour garantir la fiabilité en cas de perte ou de retard dans la transmission.

Exercice 6 : Contrôle de Flux - Limitation de Débit

Énoncé :
Un serveur reçoit des requêtes d'un client à un débit maximum de 10 requêtes par seconde. Le client envoie 15 requêtes en une seconde.

1. Que doit faire le serveur pour gérer ce débit excédentaire ?
2. Que se passe-t-il si le client continue à envoyer des requêtes à ce rythme ?
3. Si la latence est de 5 ms par message, quel est le temps de latence total pour 10 requêtes ?

—

Corrigé :
1. **Gestion du débit excédentaire :**
— Le serveur doit mettre en place une méthode de limitation du débit, par exemple :
  — Ignorer les requêtes supplémentaires jusqu'à ce que le débit redevienne acceptable.
  — Mettre en place une file d'attente pour stocker temporairement les requêtes excédentaires.
  — Répondre aux requêtes excédentaires avec un message d'erreur (par exemple, un code HTTP 429 - Trop de requêtes).
2. **Conséquences d'un envoi continu :**

— Si le client continue à envoyer des requêtes à un rythme de 15 requêtes par seconde, cela peut entraîner :
  — Un encombrement du serveur, qui peut ne pas être capable de traiter toutes les requêtes dans un délai raisonnable.
  — Un temps d'attente accru pour les requêtes en file d'attente, ce qui peut nuire à l'expérience utilisateur.
  — Un risque de refus de service si le serveur est submergé par un volume de requêtes trop élevé.

3. **Temps de latence total :**

— Si la latence est de 5 ms par message et que le serveur traite un maximum de 10 requêtes par seconde, le temps de latence total pour ces 10 requêtes est :

$$\text{Temps total} = 10 \times 5\,\text{ms} = 50\,\text{ms}.$$

— Cela signifie que, dans des conditions optimales, le serveur prend 50 ms pour traiter entièrement les 10 requêtes autorisées.

—

Conclusion : Pour gérer un débit excédentaire, un serveur doit mettre en œuvre des mécanismes de limitation tels que des files d'attente ou des messages d'erreur, tout en calculant soigneusement les temps de latence pour maintenir une expérience utilisateur acceptable. Si les limites ne sont pas respectées, le serveur risque d'être submergé, entraînant un risque de dégradation ou de refus de service.

Exercice 7

Énoncé :

Considérez deux processus, P_1 et P_2, qui doivent échanger des messages en utilisant une communication synchrone avec rendez-vous. Les deux processus doivent suivre les étapes suivantes :

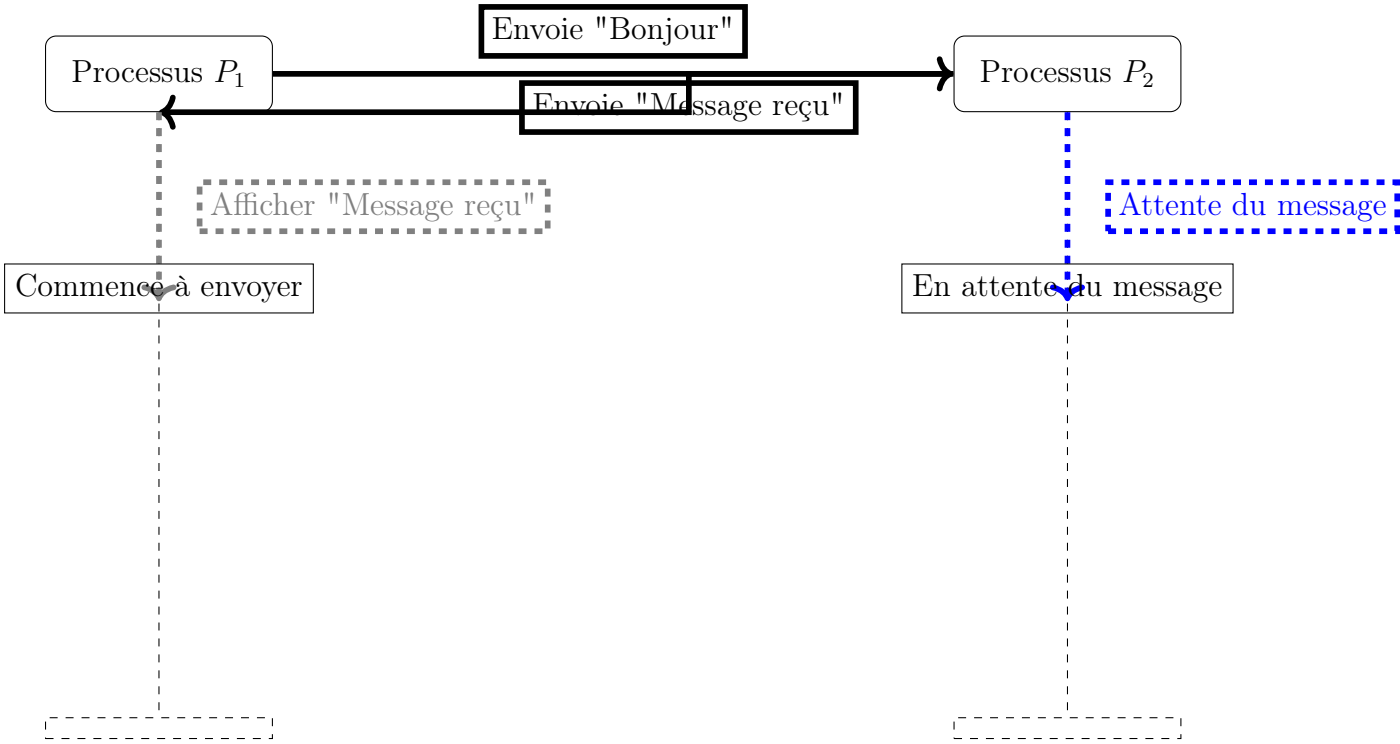
1. P_1 doit envoyer un message "Bonjour" à P_2.
2. P_2 doit attendre de recevoir le message et ensuite envoyer un accusé de réception "Message reçu" à P_1.
3. P_1 doit alors afficher l'accusé de réception reçu.

Questions :

— 1. Représentez la séquence d'échanges entre P_1 et P_2 à l'aide d'un diagramme de séquence.
— 2. Écrivez le pseudo-code pour les deux processus.

Corrigé

Réponse 1 : Diagramme de séquence



Réponse 2 : Pseudo-code

Processus P_1

```
début
    envoyer("Bonjour", P_2)
    recevoir(message_recu, P_2)
    afficher(message_recu)
fin
```

Processus P_2

```
début
    recevoir(message_envoye, P_1)
    envoyer("Message reçu", P_1)
fin
```

Éxercice 8

Ennoncé Considérez un réseau qui utilise une politique de gestion de la file d'attente FIFO (First In, First Out). Dans ce réseau, les paquets de données arrivent à un commutateur selon un certain débit et sont envoyés à destination dans l'ordre dans lequel ils ont été reçus.

1. **Arrivée des paquets** : Les paquets arrivent au commutateur selon la séquence suivante :
   — Paquet A : Arrive à $t = 0$ ms
   — Paquet B : Arrive à $t = 10$ ms
   — Paquet C : Arrive à $t = 20$ ms

— Paquet D : Arrive à $t = 30$ ms

2. **Temps de transmission** : Le temps de transmission de chaque paquet est de 15 ms.
3. **Questions** :
   (a) À quel moment chaque paquet sera-t-il envoyé ?
   (b) Quelle est la latence pour chaque paquet ?
   (c) Quel est le temps d'attente total pour chaque paquet dans le commutateur ?

Corrigé

1. Moment d'envoi de chaque paquet

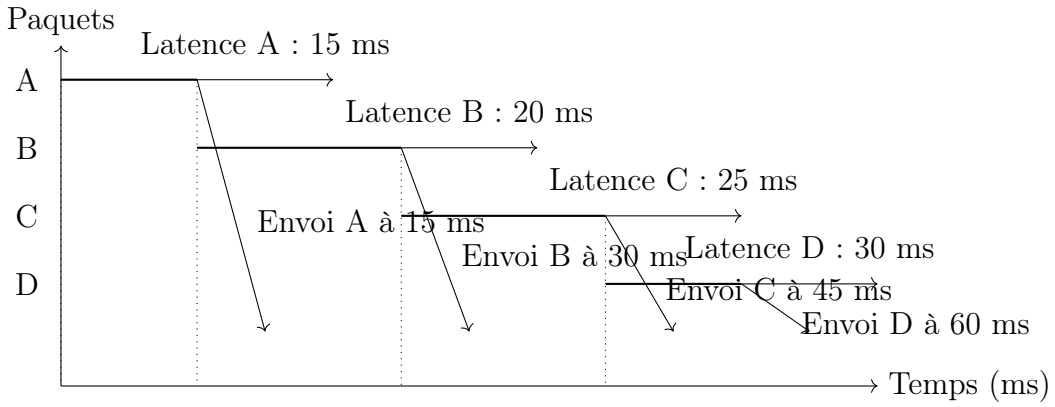
— **Paquet A** : Arrive à $t = 0$ ms, envoyé à $t = 0 + 15 = 15$ ms.
— **Paquet B** : Arrive à $t = 10$ ms, envoyé à $t = 15 + 15 = 30$ ms.
— **Paquet C** : Arrive à $t = 20$ ms, envoyé à $t = 30 + 15 = 45$ ms.
— **Paquet D** : Arrive à $t = 30$ ms, envoyé à $t = 45 + 15 = 60$ ms.

2. Latence pour chaque paquet

— **Paquet A** : $15 - 0 = 15$ ms
— **Paquet B** : $30 - 10 = 20$ ms
— **Paquet C** : $45 - 20 = 25$ ms
— **Paquet D** : $60 - 30 = 30$ ms

3. Temps d'attente total pour chaque paquet dans le commutateur

— **Paquet A** : 0 ms
— **Paquet B** : $15 - 10 = 5$ ms
— **Paquet C** : $30 - 20 = 10$ ms
— **Paquet D** : $45 - 30 = 15$ ms



Chapitre 11

Exercices sur le Temps

Exercice 1 : Synchronisation des Horloges

Énoncé : Expliquez le problème de la synchronisation des horloges dans un système distribué. Quelles sont les conséquences d'une horloge non synchronisée ?

Corrigé : La synchronisation des horloges est cruciale pour le bon fonctionnement des systèmes distribués, car elle permet de garantir que tous les processus ont une vue cohérente du temps. Une horloge non synchronisée peut entraîner des erreurs telles que des conflits dans les transactions, des incohérences dans les journaux d'événements, et des difficultés dans l'ordonnancement des tâches. Les méthodes courantes de synchronisation incluent le protocole de Berkeley et le protocole NTP (Network Time Protocol).

Exercice 2 : Mesure du Temps de Traitement

Énoncé : Un processus A prend 50 ms pour traiter une requête, tandis qu'un processus B prend 75 ms. Si A envoie une requête à B et que B traite cette requête immédiatement, quel est le temps total pris par A pour envoyer la requête, la traiter, et recevoir la réponse de B, si on suppose qu'il n'y a pas de latence de réseau ?

—

Corrigé : Le temps total pris par A pour effectuer l'opération complète inclut : - Le temps de traitement de la requête par A (50 ms). - Le temps de traitement de la requête par B (75 ms). - La latence liée à la communication entre A et B.

Si une latence de 5 ms s'applique pour chaque message échangé (envoi et réception), le temps de latence total est :

$$\text{Latence totale} = 2 \times \text{Latence par message} = 2 \times 5\,\text{ms} = 10\,\text{ms}.$$

Le temps total peut être calculé comme suit :

$$\text{Temps total} = \text{Temps A} + \text{Temps B} + \text{Latence totale}.$$

En remplaçant par les valeurs données :

$$\text{Temps total} = 50\,\text{ms} + 75\,\text{ms} + 10\,\text{ms} = 135\,\text{ms}.$$

—

Conclusion : Le temps total pris par le processus A pour envoyer la requête, la traiter et recevoir la réponse est 135 ms, en incluant la latence de communication.

Exercice 3 : Ordonnancement des Tâches

Énoncé : Supposez que trois tâches doivent être exécutées dans un système distribué. La tâche 1 prend 30 ms, la tâche 2 prend 20 ms, et la tâche 3 prend 50 ms. Si ces tâches doivent être exécutées séquentiellement, quel sera le temps d'exécution total ?

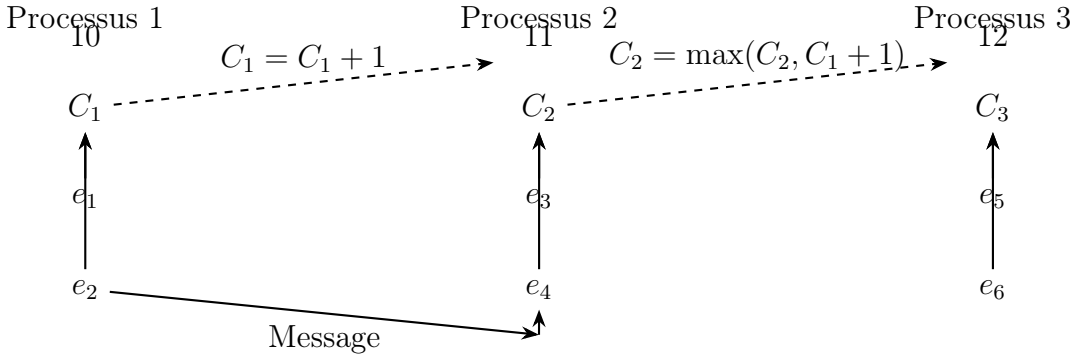
Corrigé : Le temps d'exécution total est la somme des temps d'exécution de chaque tâche.

$$\text{Temps total} = 30 \text{ ms} + 20 \text{ ms} + 50 \text{ ms} = 100 \text{ ms}$$

Exercice 4 : Algorithme de Lamport

Énoncé : Décrivez brièvement l'algorithme de Lamport pour la gestion des horloges logiques. Pourquoi est-il nécessaire dans les systèmes distribués ?

Corrigé : L'algorithme de Lamport utilise des horloges logiques pour ordonner les événements dans un système distribué. Chaque processus maintient un compteur qui est incrémenté à chaque événement local. Lorsqu'un processus envoie un message, il inclut la valeur de son compteur. À la réception d'un message, le processus met à jour son compteur pour être supérieur à la valeur reçue, assurant ainsi un ordre total des événements. Cela permet de résoudre les problèmes de causalité dans un environnement distribué.



Exercice 5 : Calcul du Temps d'Exécution dans un Réseau

Énoncé : Un message envoyé d'un processus A à un processus B prend 10 ms pour le transfert. Si B traite ce message pendant 40 ms avant de répondre, quel est le temps total avant qu'A reçoive la réponse ?

Corrigé : Le temps total avant que A reçoive la réponse est la somme du temps de transfert et du temps de traitement :

$$\text{Temps total} = \text{Temps de transfert} + \text{Temps de traitement} = 10 \text{ ms} + 40 \text{ ms} = 50 \text{ ms}$$

Exercice 6 : Temps de Latence et Débit

Énoncé : Un système distribué a un temps de latence de 5 ms et un débit de 1000 messages par seconde. Calculez le temps nécessaire pour envoyer 10 messages en tenant compte de la latence.

Corrigé : Le temps nécessaire pour envoyer 10 messages comprend la latence et le temps de transmission des messages. Le temps de transmission pour chaque message est donné par l'inverse du débit :

$$\text{Temps de transmission par message} = \frac{1}{1000 \text{ messages/s}} = 1 \text{ ms}$$

Ainsi, le temps total pour envoyer 10 messages est :

Temps total = Latence+(Nombre de messages×Temps de transmission par message) = 5 ms+(10×1 m

Exercice 7

Enoncé : Considérez un système où plusieurs processus s'exécutent dans un environnement synchrone. Chaque processus doit attendre la fin d'un événement (signal) avant de continuer son exécution.

1. Définissez ce qu'est un environnement synchrone et comment il diffère d'un environnement asynchrone.
2. Supposons que trois processus P_1, P_2 et P_3 doivent s'exécuter dans cet environnement. Chaque processus a un temps d'exécution donné :
   — P_1 : 5 ms
   — P_2 : 3 ms
   — P_3 : 4 ms

   Chaque processus doit attendre que le précédent ait terminé son exécution avant de commencer.
3. Représentez le comportement des processus sur un diagramme de séquence. Indiquez les temps d'attente et d'exécution.

Corrigé

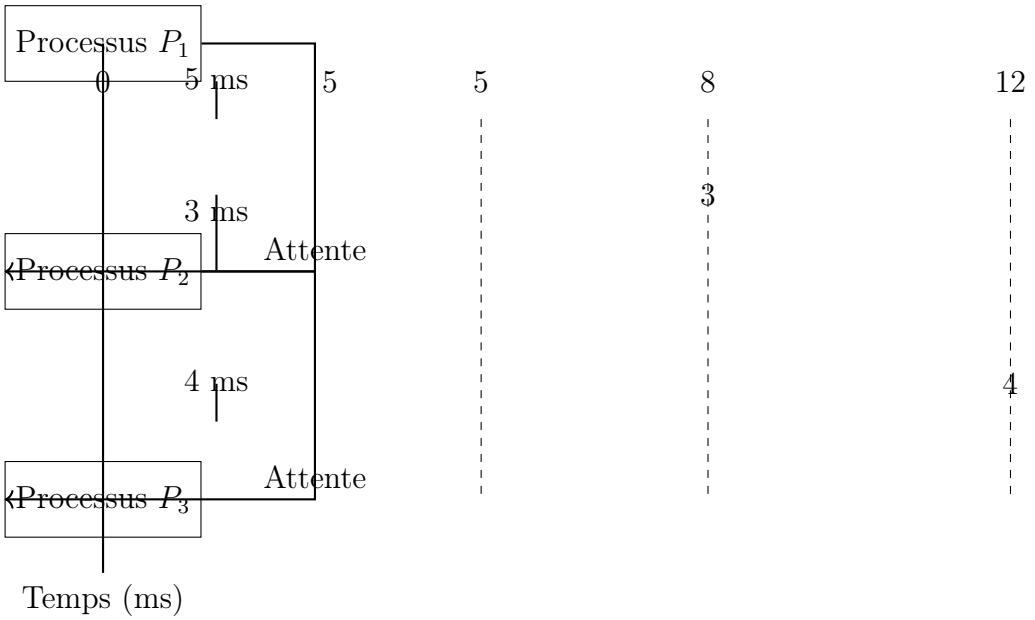
1. Définition d'un environnement synchrone

Un environnement synchrone est un cadre d'exécution où les processus sont coordonnés temporellement, ce qui signifie que chaque processus doit attendre que certains événements se produisent (comme la fin d'un autre processus) avant de pouvoir continuer. Dans un environnement asynchrone, les processus peuvent s'exécuter indépendamment les uns des autres, sans nécessité d'attendre des événements.

2. Calcul du temps d'exécution total

- Temps d'exécution de P_1 : 5 ms - Temps d'exécution de P_2 : 3 ms - Temps d'exécution de P_3 : 4 ms

3. Diagramme de Séquence



Exercice 8

Enoncé : Considérez un système de traitement de données en temps réel, où plusieurs processus doivent synchroniser leurs opérations en fonction d'un temps physique. Chaque processus doit effectuer une tâche qui prend un certain temps et doit respecter un temps d'exécution total.

1. Définissez ce qu'est le temps physique dans un environnement synchrone et son importance dans le contexte des systèmes distribués.
2. Supposons que quatre processus P_1, P_2, P_3, et P_4 doivent s'exécuter dans cet environnement. Chaque processus a un temps d'exécution donné :
   — P_1 : 4 ms
   — P_2 : 6 ms
   — P_3 : 5 ms
   — P_4 : 3 ms
   Chaque processus doit attendre que le précédent ait terminé son exécution avant de commencer.
3. Représentez le comportement des processus sur un diagramme de séquence. Indiquez les temps d'attente et d'exécution.

Corrigé

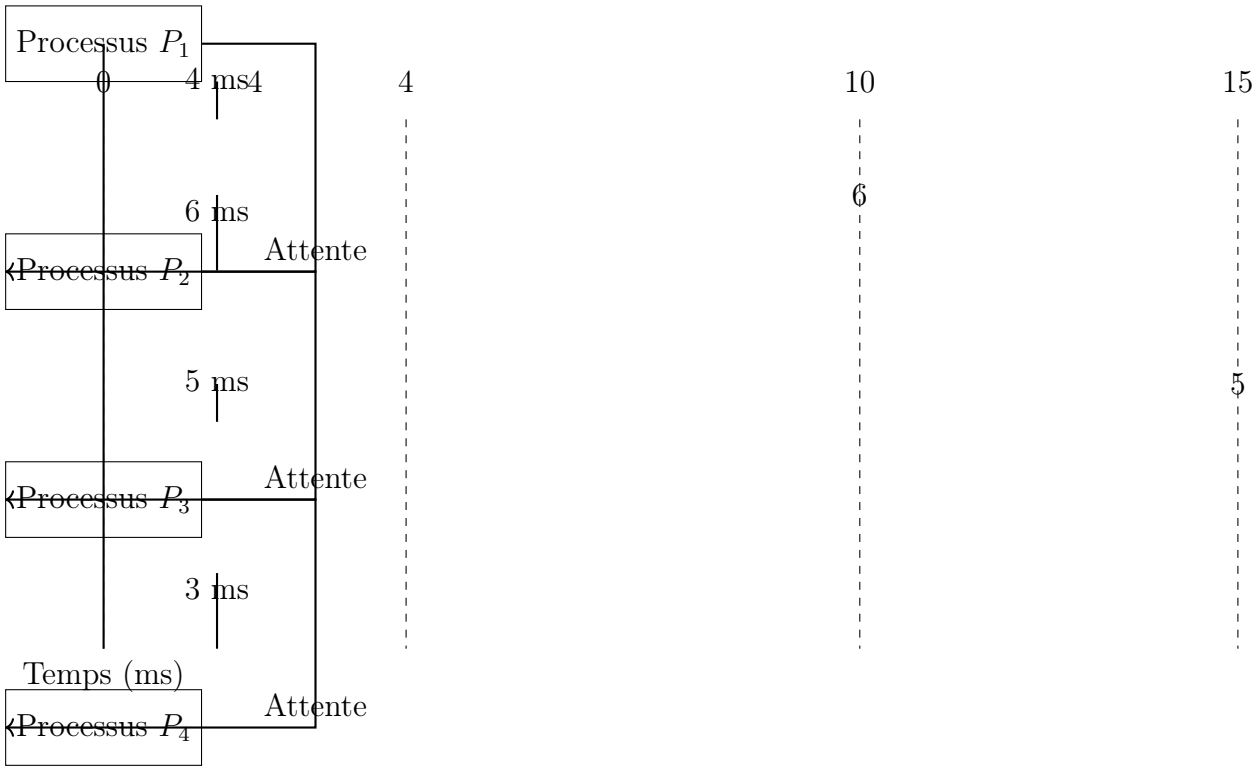
1. Définition du temps physique

Le temps physique dans un environnement synchrone est la mesure de l'écoulement du temps qui est utilisée pour synchroniser les opérations entre les différents processus. Il est essentiel dans les systèmes distribués pour garantir que les actions des processus sont coordonnées et se produisent dans un ordre défini, ce qui est crucial pour la cohérence des données et le respect des délais.

2. Calcul du temps d'exécution total

- Temps d'exécution de P_1 : 4 ms - Temps d'exécution de P_2 : 6 ms - Temps d'exécution de P_3 : 5 ms - Temps d'exécution de P_4 : 3 ms

3. Diagramme de Séquence



Exercice 9

Enoncé Considérez un système distribué utilisant des horloges physiques pour synchroniser les processus. Chaque processus doit effectuer des tâches à des intervalles spécifiques en se basant sur des horloges synchronisées.

1. Définissez ce qu'est une horloge physique dans un environnement synchrone et son rôle dans la synchronisation des processus.
2. Supposons que trois processus P_1, P_2, et P_3 doivent s'exécuter en utilisant des horloges physiques. Les temps d'exécution et les intervalles de démarrage sont les suivants :
   — P_1 : 3 ms (démarre à 0 ms)
   — P_2 : 5 ms (démarre à 3 ms)
   — P_3 : 4 ms (démarre à 8 ms)
   Chaque processus doit respecter son intervalle d'exécution.
3. Représentez le comportement des processus sur un diagramme de séquence. Indiquez les temps de démarrage et d'exécution.

Corrigé

1. **Définition d'une horloge physique :**

   Une horloge physique dans un environnement synchrone est un dispositif matériel ou logiciel utilisé pour mesurer le temps dans un système distribué. Elle permet aux processus de synchroniser leurs actions en se basant sur une notion commune du temps. Les horloges physiques jouent un rôle crucial dans la synchronisation des processus en garantissant que les événements critiques se produisent dans l'ordre correct, même si les processus s'exécutent sur des machines distinctes.

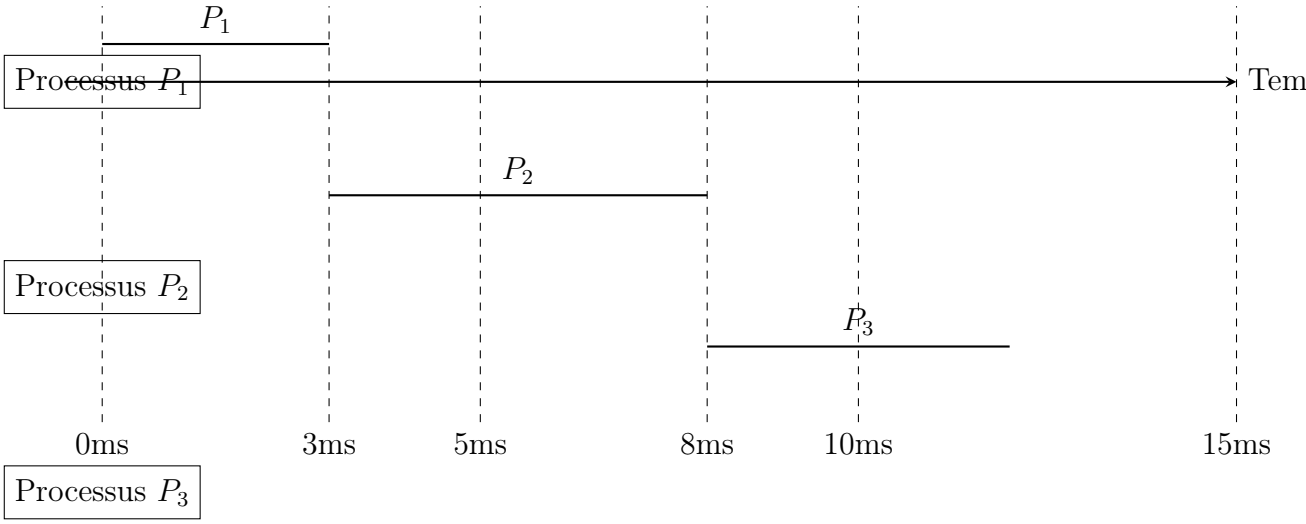
   Rôle dans la synchronisation des processus : - Les horloges physiques permettent d'exécuter des tâches périodiques à des intervalles prédéfinis. - Elles garantissent que les délais temporels sont respectés, ce qui est essentiel dans les systèmes temps réel. - Elles aident à maintenir la cohérence temporelle dans les échanges de messages et l'exécution des tâches.

2. **Exécution des processus :**

   Voici les temps de démarrage et d'exécution de chaque processus :
   — P_1 : Démarre à 0 ms, exécute pendant 3 ms.
   — P_2 : Démarre à 3 ms, exécute pendant 5 ms.
   — P_3 : Démarre à 8 ms, exécute pendant 4 ms.

3. **Diagramme de séquence :**

   Le comportement des processus peut être représenté par le diagramme de séquence suivant :



   Description : - P_1 commence à 0 ms et s'exécute jusqu'à 3 ms. - P_2 commence à 3 ms et s'exécute jusqu'à 8 ms. - P_3 commence à 8 ms et s'exécute jusqu'à 12 ms.

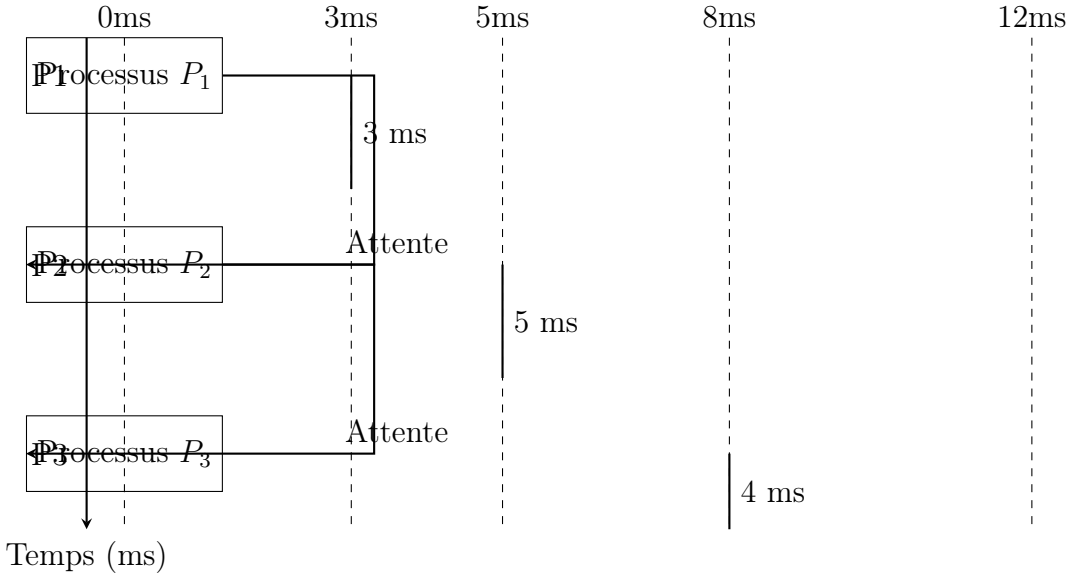
1. Définition des horloges physiques

Une horloge physique dans un environnement synchrone est un dispositif qui mesure le temps écoulé et permet aux processus d'être synchronisés en fonction de cette mesure. Elle joue un rôle crucial en fournissant un cadre temporel commun pour la coordination des opérations dans un système distribué, garantissant que les processus s'exécutent à des moments précis.

2. Temps d'exécution des processus

- Temps d'exécution de P_1 : 3 ms (démarre à 0 ms) - Temps d'exécution de P_2 : 5 ms (démarre à 3 ms) - Temps d'exécution de P_3 : 4 ms (démarre à 8 ms)

3. Diagramme de Séquence



Chapitre 12

Exercices sur les Algorithmes de Concurrence

Exercice 1 : Protocole de Mutual Exclusion

Énoncé : Décrivez un protocole de mutual exclusion utilisant l'algorithme de Lamport. Indiquez les étapes pour entrer et sortir de la section critique.

Corrigé : L'algorithme de Lamport pour la mutual exclusion fonctionne comme suit :

— Chaque processus envoie une demande pour entrer dans la section critique, incluant son horodatage.

— Les processus reçoivent les demandes et les stockent dans une file d'attente.

— Un processus peut entrer dans la section critique si sa demande est la plus ancienne dans la file d'attente et si aucun autre processus ne se trouve dans la section critique.

— Lorsqu'il sort, le processus notifie les autres de sa sortie.

—

Exercice 2 : Problème des Philosophes

Énoncé : Modélisez le problème des philosophes mangeurs en utilisant des sémaphores. Expliquez comment éviter les interblocages.

Corrigé :

— Chaque philosophe est représenté par un processus.

— Les fourchettes sont représentées par des sémaphores binaires.

— Pour éviter les interblocages, un philosophe doit prendre les fourchettes dans un ordre fixe (par exemple, toujours la fourchette gauche avant la droite) et les relâcher immédiatement après avoir mangé.

—

Exercice 3 : Producteur-Consommateur

Énoncé : Implémentez un modèle de producteur-consommateur en utilisant des mutex et des conditions. Décrivez les étapes de la production et de la consommation.

Corrigé :

— Le producteur génère un produit et le place dans un buffer protégé par un mutex.

— Si le buffer est plein, le producteur se bloque jusqu'à ce qu'un consommateur consomme un produit.
— Le consommateur récupère un produit du buffer, libère le mutex et notifie le producteur si le buffer n'est plus plein.

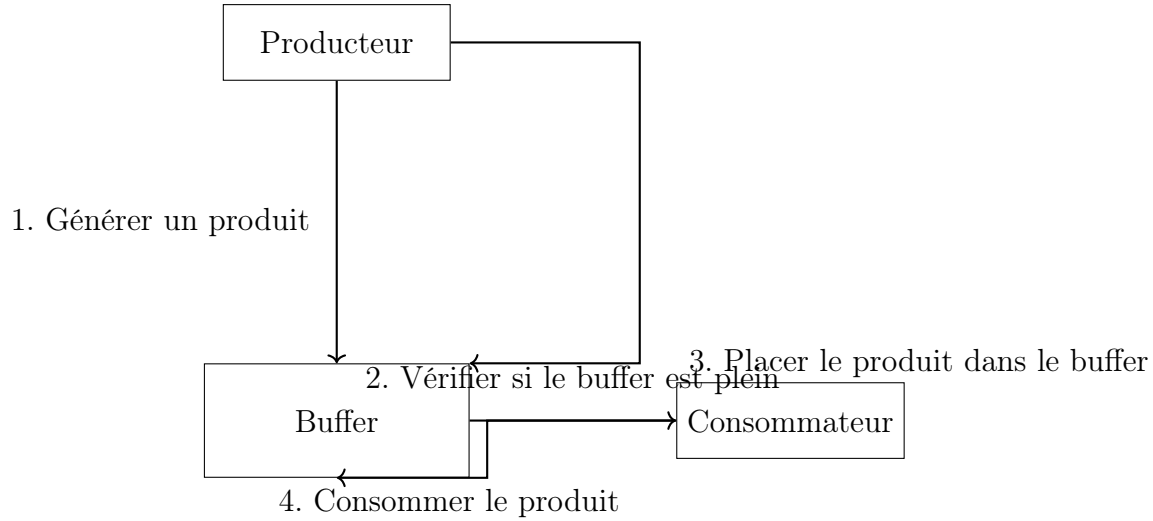


FIGURE 12.1 – Schéma explicatif du modèle Producteur-Consommateur

—

Exercice 4 : Système de Fichiers Concurrent

Énoncé : Concevez un système de fichiers concurrent où plusieurs processus peuvent lire et écrire des fichiers simultanément. Décrivez comment vous utiliseriez des sémaphores.

Corrigé :
— Utilisez un sémaphore pour contrôler l'accès en écriture et un autre pour le contrôle de lecture.
— Un processus qui écrit doit acquérir un sémaphore d'écriture et empêcher les lectures.
— Les processus de lecture acquièrent un sémaphore de lecture, permettant plusieurs lectures simultanées tant qu'aucune écriture n'est en cours.
—

Exercice 5 : Algorithmes de Synchronisation

Énoncé : Décrivez le concept de synchronisation dans les systèmes concurrents. Quelles techniques pouvez-vous utiliser pour synchroniser des processus ?

Corrigé : La synchronisation est essentielle pour coordonner l'exécution des processus afin d'éviter les conflits d'accès aux ressources partagées. Les techniques de synchronisation incluent :
— Mutex : pour un accès exclusif à une ressource.
— Sémaphores : pour gérer l'accès à un nombre limité de ressources.

— Moniteurs : pour encapsuler la synchronisation et la gestion de ressources partagées dans un objet.

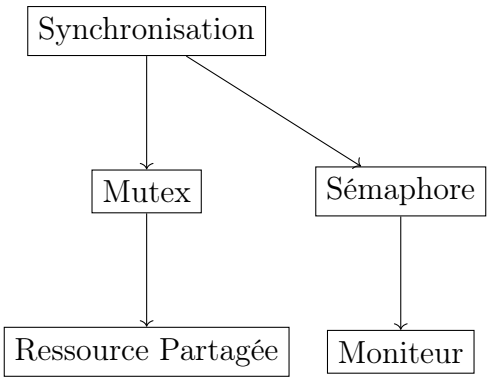


FIGURE 12.2 – Schéma explicatif de la synchronisation dans les systèmes concurrents

Exercice 6

Enoncé : L'algorithme de Lamport est utilisé pour synchroniser les horloges dans un système distribué. Considérez un système composé de trois processus P_1, P_2 et P_3, qui échangent des messages pour maintenir un ordre total des événements.

1. Définissez brièvement l'algorithme de Lamport et son objectif principal.
2. Supposons que les processus échangent des messages selon le schéma suivant :
   — P_1 envoie un message à P_2.
   — P_2 envoie un message à P_3.
   — P_3 envoie un message à P_1.
   Montrez les horloges locales de chaque processus après chaque opération et indiquez l'ordre des messages échangés.
3. Représentez le comportement des processus sur un diagramme de séquence.

Corrigé

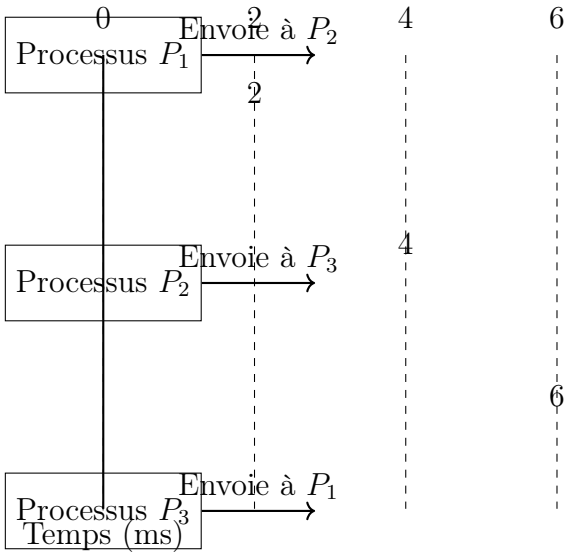
1. Définition de l'algorithme de Lamport

L'algorithme de Lamport est une méthode utilisée pour synchroniser les horloges logiques dans un système distribué. Son objectif principal est de garantir un ordre total des événements en attribuant des timestamps (horodatages) aux messages échangés entre les processus. Chaque processus maintient un compteur local qui est incrémenté à chaque événement interne ou lors de l'envoi d'un message.

2. Échange de messages et horloges locales

Processus	Action	Horloge Avant	Horloge Après	Timestamp
P_1	Envoie un message à P_2	1	2	2
P_2	Reçoit le message de P_1	1	3	2
P_2	Envoie un message à P_3	3	4	4
P_3	Reçoit le message de P_2	1	5	4
P_3	Envoie un message à P_1	5	6	6
P_1	Reçoit le message de P_3	2	7	6

3. Diagramme de Séquence



Exercice 7

L'algorithme de Ricart et Agrawala est un protocole distribué pour gérer l'accès aux ressources critiques dans un système distribué. Considérez un système composé de trois processus P_1, P_2 et P_3 qui souhaitent accéder à une ressource partagée.

1. Définissez brièvement l'algorithme de Ricart et Agrawala et son objectif principal.
2. Supposons que les processus échangent des messages pour demander l'accès à la ressource partagée selon le schéma suivant :
   — P_1 demande l'accès à la ressource.
   — P_2 et P_3 répondent à P_1.
   — P_1 accède à la ressource.

   Montrez les horloges locales de chaque processus après chaque opération et indiquez l'ordre des messages échangés.
3. Représentez le comportement des processus sur un diagramme de séquence.

Corrigé

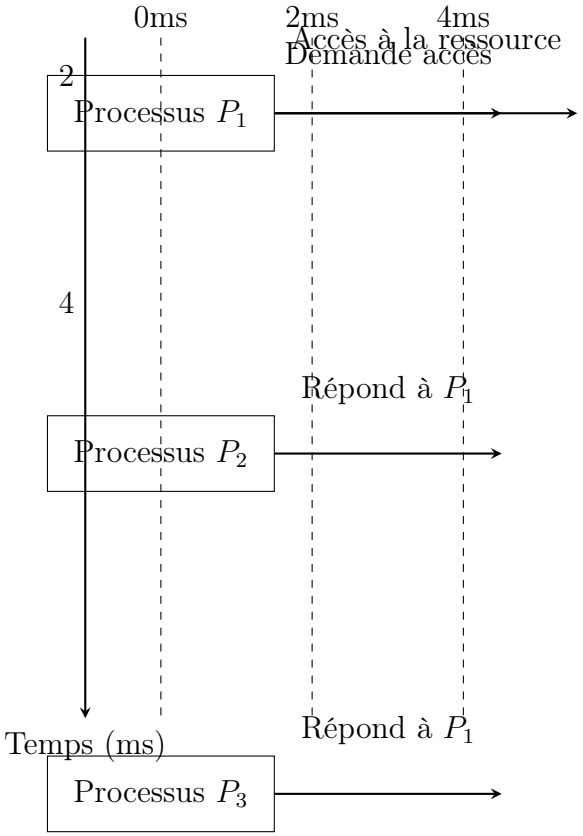
1. Définition de l'algorithme de Ricart et Agrawala

L'algorithme de Ricart et Agrawala est un protocole distribué utilisé pour synchroniser l'accès à une ressource critique dans un système distribué. Son objectif principal est de garantir qu'un processus ne peut accéder à la ressource qu'après avoir reçu des réponses positives de tous les autres processus. Cela permet d'éviter les conflits et de maintenir l'intégrité des données.

2. Échange de messages et horloges locales

Processus	Action	Horloge Avant	Horloge Après	Timestamp
P_1	Demande l'accès à la ressource	1	2	2
P_2	Reçoit la demande de P_1 et répond	1	2	2
P_3	Reçoit la demande de P_1 et répond	1	2	2
P_1	Accède à la ressource	2	3	-

3. Diagramme de Séquence



Exercice 8

L'algorithme de Carvalho et Roucairol est un protocole distribué pour gérer l'accès à une ressource critique dans un système distribué. Considérez un système composé de quatre processus P_1, P_2, P_3 et P_4 qui souhaitent accéder à une ressource partagée.

1. Définissez brièvement l'algorithme de Carvalho et Roucairol et son objectif principal.
2. Supposons que les processus échangent des messages pour demander l'accès à la ressource partagée selon le schéma suivant :
   — P_1 demande l'accès à la ressource.
   — P_2 et P_3 répondent à P_1.
   — P_1 accède à la ressource.
   — P_4 demande également l'accès à la ressource après que P_1 a terminé.

Montrez les horloges locales de chaque processus après chaque opération et indiquez l'ordre des messages échangés.

3. Représentez le comportement des processus sur un diagramme de séquence.

Corrigé

1. Définition de l'algorithme de Carvalho et Roucairol

L'algorithme de Carvalho et Roucairol est un protocole distribué utilisé pour gérer l'accès aux ressources critiques dans un système distribué. Son objectif principal est d'assurer que les processus accèdent à la ressource partagée de manière ordonnée et sans conflits, en garantissant que seul un processus à la fois puisse accéder à la ressource.

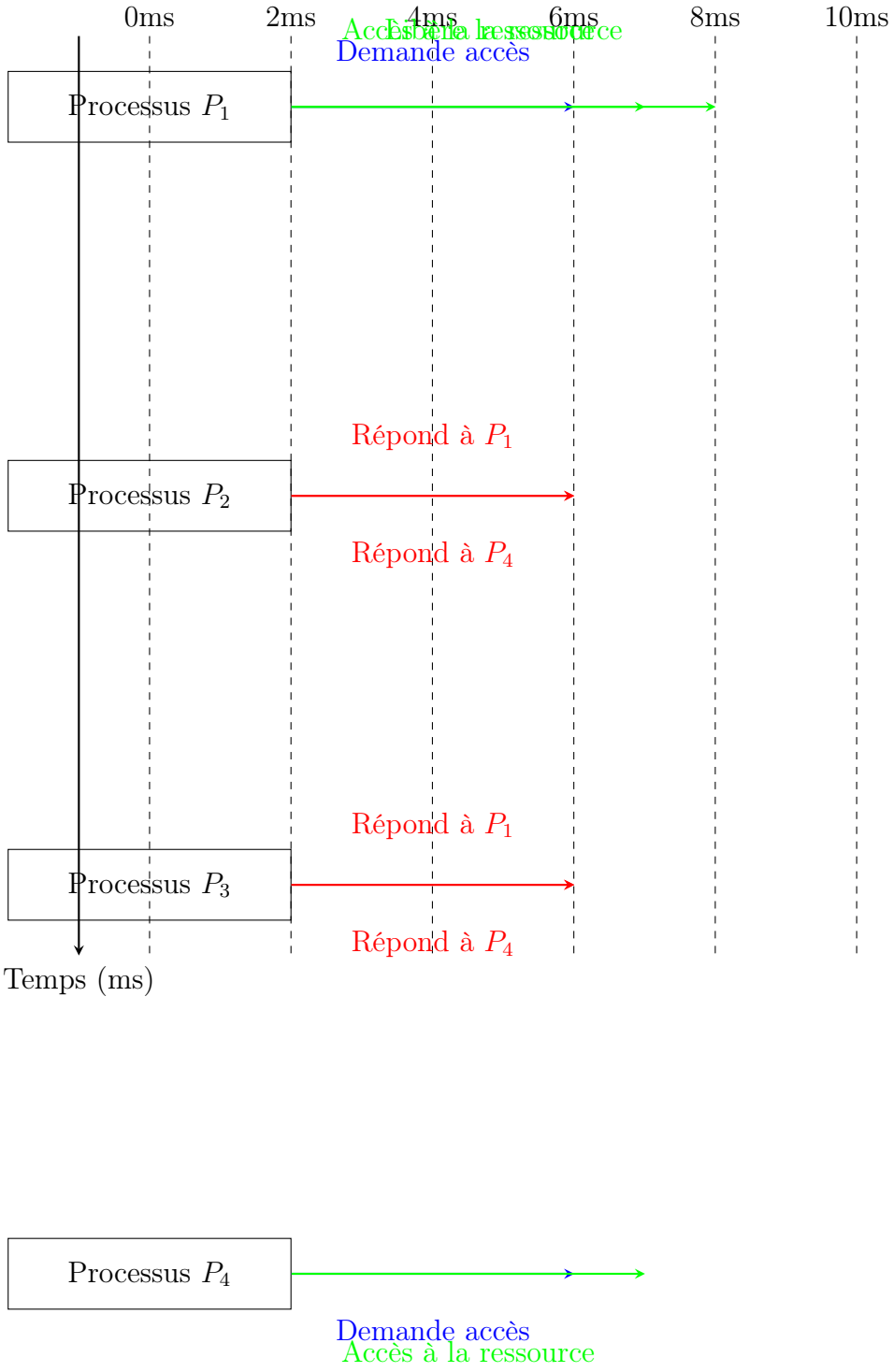
2. Échange de messages et horloges locales

Processus	Action	Horloge Avant	Horloge Après	Timestamp
P_1	Demande l'accès à la ressource	1	2	2
P_2	Reçoit la demande de P_1 et répond	1	2	2
P_3	Reçoit la demande de P_1 et répond	1	2	2
P_1	Accède à la ressource	2	3	-
P_1	Libère la ressource	3	4	-
P_4	Demande l'accès à la ressource	1	2	2
P_2	Répond à P_4	2	3	3
P_3	Répond à P_4	2	3	3
P_4	Accède à la ressource	2	3	-

3. Diagramme de Séquence

Processus	Action	Horloge Avant	Horloge Après	Timestamp
P_1	Demande l'accès à la ressource	1	2	2
P_2	Reçoit la demande de P_1 et répond	1	2	2
P_3	Reçoit la demande de P_1 et répond	1	2	2
P_1	Accède à la ressource	2	3	-
P_1	Libère la ressource	3	4	-
P_4	Demande l'accès à la ressource	1	2	2
P_2	Répond à P_4	2	3	3
P_3	Répond à P_4	2	3	3
P_4	Accède à la ressource	2	3	-

3. Diagramme de Séquence



Chapitre 13

Exercices sur l'Observation

Exercice 1 : Observation des Systèmes Distribués

Énoncé : Considérez un système distribué composé de trois nœuds (A, B et C) qui communiquent entre eux. Chaque nœud enregistre des événements. Établissez un tableau pour représenter les événements observés par chaque nœud et leur ordre d'occurrence.

Corrigé :

Nœud	Événements observés
A	e1, e2, e3
B	e2, e4, e5
C	e1, e3, e5

TABLE 13.1 – Événements observés par chaque nœud.

Ordre d'occurrence :

1. e1
2. e2
3. e3
4. e4
5. e5

—

Explication de l'ordre d'occurrence :

L'ordre d'occurrence des événements dans un système distribué peut être déterminé en tenant compte des relations causales entre les événements, c'est-à-dire l'ordre dans lequel ces événements se produisent par rapport aux autres. Voici comment cet ordre a été déterminé :

1. **Événement e1 :** L'événement $e1$ est observé par les nœuds A et C. Comme $e1$ est observé en premier par ces nœuds, il est donc le premier événement à se produire.
2. **Événement e2 :** L'événement $e2$ est observé par A et B. Comme $e2$ est observé par A après $e1$, cet événement doit se produire après $e1$. Par conséquent, $e2$ est le deuxième événement dans l'ordre d'occurrence.
3. **Événement e3 :** L'événement $e3$ est observé par A et C. Il doit se produire après $e2$, car A observe $e2$ avant $e3$, et C observe $e3$ après $e2$. Ainsi, $e3$ se produit après $e2$ et avant $e4$ et $e5$.

4. **Événement e4 :** L'événement $e4$ est observé uniquement par B. Comme B observe $e4$ après $e3$, cet événement se produit après $e3$, et il est placé en quatrième position dans l'ordre d'occurrence.
5. **Événement e5 :** L'événement $e5$ est observé à la fois par B et C. Puisque cet événement est observé après $e4$, il est placé en dernière position dans l'ordre des événements.

Ainsi, l'ordre des événements, basé sur les relations causales entre les différentes observations des nœuds, est le suivant :

1. e1
2. e2
3. e3
4. e4
5. e5

Exercice 2 : Horloges Logiques

Énoncé : Supposons que vous ayez les horloges logiques des nœuds suivants dans un système distribué :
— Nœud A : 3
— Nœud B : 5
— Nœud C : 2
Un message est envoyé de A à B et de B à C. Quel sera l'état des horloges logiques après ces événements ?
Corrigé :
— Nœud A envoie à B : A = 3 → B = max(5, 3 + 1) = 6
— Nœud B envoie à C : B = 6 → C = max(2, 6 + 1) = 7
État final des horloges logiques :
— Nœud A : 4 (après envoi)
— Nœud B : 7 (après réception)
— Nœud C : 7 (après réception)
—

Exercice 3 : Équivalence des Observations

Énoncé : Définissez et expliquez ce qu'on entend par « équivalence des observations » dans un système distribué. Donnez un exemple illustratif.
Corrigé : L'équivalence des observations fait référence à la capacité des nœuds d'un système distribué à observer et à traiter des événements de manière cohérente. Cela implique que les nœuds doivent être en mesure de parvenir à un consensus sur l'ordre des événements, même si les messages sont reçus dans un ordre différent.
Exemple : Si le nœud A observe l'événement e1 avant e2 et que le nœud B observe e2 avant e1, tant que les actions résultantes de ces événements n'interfèrent pas l'une avec l'autre, l'état final du système reste équivalent.
—

Exercice 4 : Protocole d'Observation

Énoncé : Considérez un protocole d'observation utilisé dans un système de capteurs. Expliquez le rôle de chaque élément suivant :

— a) Capteur
— b) Observation
— c) Rapport

Corrigé : a) **Capteur :** Dispositif qui collecte des données d'un environnement spécifique (par exemple, température, humidité).

b) **Observation :** Processus de collecte de données par le capteur à un moment donné. C'est l'enregistrement des valeurs mesurées par le capteur.

c) **Rapport :** Ensemble d'informations collectées à partir des observations, souvent structuré pour une analyse ultérieure. Les rapports permettent de synthétiser les données et de fournir des insights sur le comportement du système.

Chapitre 14

Exercices sur les Algorithmes d'Élection

Exercice 1

Énoncé : Considérez un système distribué avec 5 processus numérotés de 1 à 5. Chaque processus peut communiquer avec tous les autres. Utilisez l'algorithme de l'élection de Ring pour déterminer le leader. Décrivez les étapes de l'algorithme.

Corrigé

1. Chaque processus envoie son identifiant à son voisin. 2. Un processus qui reçoit un identifiant plus grand que le sien le transmet au voisin suivant. 3. Lorsque le processus reçoit son propre identifiant, il devient le leader.

Exercice 2

Énoncé : Dans un système avec 4 processus, chacun ayant un identifiant unique. Expliquez comment l'algorithme de l'élection de Bully fonctionne.

Corrigé

1. Un processus initie une élection en envoyant un message d'élection aux processus ayant un identifiant supérieur. 2. Si un processus reçoit un message d'élection et a un identifiant supérieur, il répond par un message « je suis en vie » et initie une nouvelle élection. 3. Si un processus ne reçoit pas de réponse, il devient le leader.

Exercice 3

Énoncé : Implémentez un pseudo-code pour l'algorithme de l'élection de Ring.

Corrigé

```
procedure ringElection(currentProcess)
    send ID to nextProcess
```

```
    receive message from previousProcess
    if message is myID then
        declare myself as leader
    else if message > myID then
        send message to nextProcess
    end if
end procedure
```

Exercice4

Énoncé : Quelles sont les implications de l'échec d'un leader sur l'algorithme d'élection de Bully ?

Corrigé

- Si un leader échoue, un nouvel algorithme d'élection doit être initié. - Les processus doivent détecter l'absence de réponse du leader pour relancer l'élection.

Chapitre 15

Exercices sur La mémoire virtuelle répartie et linéarisabilité

Exercices

Exercice 1 : Concepts de Base

Énoncé : Définissez les termes suivants :

— Mémoire virtuelle répartie

— Linéarisabilité

Corrigé :

— **Mémoire virtuelle répartie :** Une technique permettant à un système de distribuer l'espace mémoire sur plusieurs machines, donnant l'illusion d'une mémoire unique aux utilisateurs.

— **Linéarisabilité :** Une propriété des systèmes distribués qui garantit que les opérations d'un objet partagé apparaissent comme si elles avaient été exécutées instantanément à un certain moment dans le temps.

Exercice 2 : Identification de la Linéarisabilité

Énoncé : Considérez les opérations suivantes effectuées sur une mémoire partagée : 1. 'write(A, 5)' 2. 'read(A) → x' 3. 'write(A, 10)' 4. 'read(A) → y'

Déterminez si la séquence est linéarisable.

Corrigé : La séquence n'est pas linéarisable si 'x' est égal à 10 après la lecture. Si 'x' est 5, alors la séquence est linéarisable.

Exercice 3 : Évaluation de la Performance

Énoncé : Quel est l'impact de l'augmentation du nombre de nœuds dans une architecture de mémoire virtuelle répartie sur la latence d'accès ?

Corrigé : L'augmentation du nombre de nœuds peut entraîner une latence accrue due à la surcharge de gestion des communications et des synchronisations entre les nœuds. Cependant, cela peut également améliorer la bande passante et la tolérance aux pannes.

Exercice 4 : Protocole de Consistance

Énoncé : Expliquez le rôle d'un protocole de consistance dans un système de mémoire virtuelle répartie.

Corrigé : Un protocole de consistance définit les règles pour garantir que tous les nœuds voient les mêmes valeurs pour un objet partagé à tout moment. Cela assure la cohérence des données et le respect des propriétés de linéarisabilité.

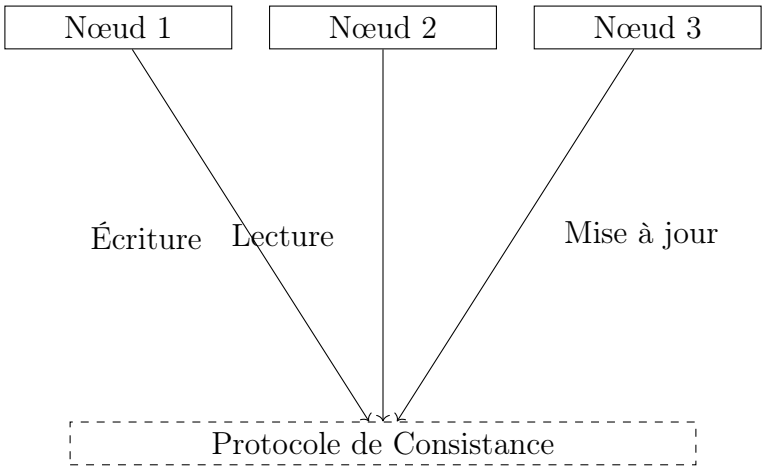


FIGURE 15.1 – Schéma d'un Protocole de Consistance dans un Système de Mémoire Virtuelle Répartie (Amélioré)

Exercice 5 : Analyse de Scénario

Énoncé : Dans un système distribué, trois processus P1, P2, et P3 effectuent des écritures et des lectures sur une mémoire partagée. Si P1 écrit 1, P2 lit 1, et P3 écrit 2, est-il possible que P2 lise 2 ?

Corrigé : Non, P2 ne peut pas lire 2 après avoir lu 1 si les opérations sont linéarisables. Cela indique que P1 doit avoir précédé l'opération de P3, et donc P2 lira soit 1 soit 2, mais pas à la fois.

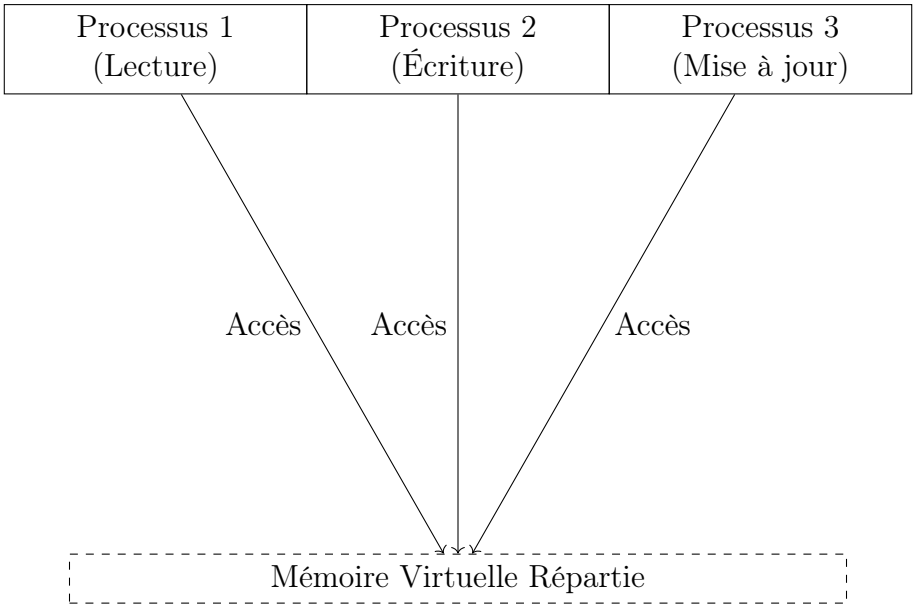


FIGURE 15.2 – Schéma de la Gestion de la Mémoire Virtuelle dans un Système Réparti (Amélioré)

Exercice 6 : Implémentation

Énoncé : Décrivez comment implémenter un tableau associatif en utilisant la mémoire virtuelle répartie. Quels défis peuvent survenir en termes de linéarisabilité ?

Corrigé : Un tableau associatif peut être implémenté en utilisant des clés pour accéder à des valeurs stockées sur différents nœuds. Les défis incluent la gestion des accès concurrents, l'application de règles de verrouillage pour garantir la linéarisabilité, et la latence due à la communication entre nœuds.

Bibliographie

[1] B. Hayat, "GRILLES DE CALCUL," 2024. univ-usto.dz

[2] L. Cudennec, "Contributions pour l'utilisation des machines parallèles, réparties et hétérogènes à travers le prisme de la gestion des données partagées," 2022. hal.science

[3] N. Yazid and N. E. Mehadji, "Transmission radio sur fibre pour les réseaux 5G," 2024. univ-tlemcen.dz

[4] P. Quentel, "Architecture multi-agent distribuée et collaborative pour l'allocation de tâches à des senseurs : application aux systèmes navals," 2024. hal.science

[5] A. K. Sophia, "Modèles de service à base d'agents mobiles pour l'amélioration de la communication et de la sécurité dans un environnement intelligent JURY," 2020. imist.ma

[6] Z. Meftah, "Une approche cloud computing basée IoT pour le smart House," 2021. univ-biskra.dz

[7] M. Méghaichi, H. Meghlaoui, and E. E. Kerkouche, "Une Approche Outillée pour l'Intégration des Processus Métier (BPMN) dans le Développement des Applications de l'Internet des Objets," 2022. univ-jijel.dz

[8] W. B. A. B. Zoungrana, "Application des algorithmes d'apprentissage automatique pour la détection de défauts de roulements sur les machines tournantes dans le cadre de l'Industrie 4.0," 2020. uqac.ca

[9] I. ZOUANEB, "Mise en place d'une méthodologie formelle pour la simulation et la validation des architectures à base de GPU," 2023. univ-tiaret.dz

[10] S. Marir, "Modélisation Formelle des Systèmes Fog : vers l'Analyse et la Validation de leur Comportement," 2022. hal.science

[11] A. Attajer, "Conception et développement d'une architecture de pilotage distribué pour améliorer la résilience opérationnelle dans les systèmes cyber-physiques de production," 2023. [Online]. Available : imist.ma

[12] S. Zxivanovich, B. Todorovic, "L'IdO pour une nouvelle ère de réseaux unifiés, de confiance zéro et de protection accrue de la vie privée," Cybersécurité des ..., 2024. [HTML]

[13] E. N. Power, "Emerson Network Power : Systèmes de baies pour datacenters Essentiels pour la Business-Critical ContinuityTM–Global Security Mag Online," 2024. [Online]. Available : globalsecuritymag.fr

[14] M. R. Seye, "Intégration d'un nouveau modèle de communication entre acteurs dans une architecture de simulation multi agent (application à deux cas d'étude du Sénégal)," 2022. [Online]. Available : hal.science

[15] M. Z. Boucetta Yasmine, "Les protocoles de transport dans un réseau de télécommunication Jury," [Online]. Available : archives.univ-biskra.dz

[16] P. Quentel, "Architecture multi-agent distribuée et collaborative pour l'allocation de tâches à des senseurs : application aux systèmes navals," 2024. [Online]. Available : hal.science

[17] F. Guennez, "Modèle d'application du problème d'ordonnancement des taches dans les systèmes distribués résolvant les Problèmes de planification de la circulation routière dans ...," 2022. [Online]. Available : univ-tebessa.dz

[18] T. Amel and A. Hadjer, "Une étude comparative des réseaux basés sur Named Data Networking et IP paradigme," 2023. [Online]. Available : univ-bba.dz

[19] K. Hafsi, "Approche distribuée basée sur un système multi-agent pour l'optimisation énergétique d'un micro-réseau de distribution DC.," 2024. [Online]. Available : hal.science

[20] T. Docquier, "Méthodologies pour l'évaluation de performances d'architectures réseaux smart grids," 2021. [Online]. Available : hal.science

[21] M. Sabeima, M. Lamolle, and A. Anghour, "Vers une plateforme sémantique pour l'apprentissage adaptatif et collaboratif en ligne," Revue Africaine de ..., 2023. [Online]. Available : episciences.org

[22] D. Bouthaina and C. Nadjeh, "Routage en temps réel dans les réseaux de capteurs sans fil avec prolongation de la durée de vie du réseau dans le cadre de l'Internet des objets.," 2024. [Online]. Available : univ-bba.dz

[23] M. Masmoudi, "Prévention des attaques de confiance en temps réel dans l'IoT social," 2023. [Online]. Available : hal.science

[24] M. A. Chalouf, "Gestion de la qualité de service et de la sécurité dans un environnement e-santé," 2020. [HTML]

[25] R. Morseli and M. S. Mokhtari, "Détection des pannes d'un système mécanique par apprentissage automatique," 2023. [Online]. Available : `https://univ-tiaret.dz`

[26] F. Amari, M. A. Samah, and R. E. Belhadef, "Utilisation et traitement des données incomplètes pour l'étude des actions de maintenance dans les systèmes industriels : modèles de régression et d'interpolation," 2021. [Online]. Available : `https://univ-jijel.dz`

[27] S. Dridi, "Étude et validation par simulation d'une stratégie de contrôle multitâches pour un onduleur quasi Z source dédié aux systèmes photovoltaïques interfacés avec le ...," 2024. [Online]. Available : `https://etsmtl.ca`

[28] P. Quentel, "Architecture multi-agent distribuée et collaborative pour l'allocation de tâches à des senseurs : application aux systèmes navals," 2024. [Online]. Available : `https://hal.science`

[29] R. Chabani and F. E. Bouchaib, *Implémentation d'un Protocole d'Élection d'Un Serveur d'Authentification dans l'Internet des Objets*, 2021. univ-jijel.dz

[30] K. BENYAHIA and A. BENGUEDDOUDJ, *Une Approche Efficace de Clustering pour Améliorer les Performances dans l'IoT, basée sur les Réseaux de Capteurs Sans Fil*, 2024. univ-bba.dz

[31] A. Messiaid, *Optimisation de la sélection des cluster heads pour améliorer l'efficacité des réseaux de capteurs sans fil*, dspace.univ-ouargla.dz. univ-ouargla.dz

[32] S. Chaima and B. Hana, *Une nouvelle approche de clustering pour les réseaux véhiculaires modernes (IoV)*, 2023. univ-bba.dz

[33] D. Bouthaina and C. Nadjeh, *Routage en temps réel dans les réseaux de capteurs sans fil avec prolongation de la durée de vie du réseau dans le cadre de l'Internet des objets*, 2024. univ-bba.dz

[34] C. CHEKHAR and M. KADDI, *Une nouvelle approche pour la minimisation de la consommation d'énergie dans les réseaux de capteurs sans fil*, 2022. univ-adrar.edu.dz

[35] A. B. E. N. CHABANE and B. E. N. CHABANE, *Collection efficace des données par puits mobile dans un réseau de capteurs sans fil*, 2024. univ-bba.dz