الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

UNIVERSITÉ BADJI MOKHTAR - ANNABA
BADJI MOKHTAR – ANNABA UNIVERSITY

جامعة باجي مختار – عنابـــة

Faculté : Des Sciences de L'ingéniorat

Département : D'Informatique

Domaine : Math et Informatique

Filière : Informatique

Spécialité : Systèmes Informatique et Décisions

Mémoire

Présenté en vue de l'obtention du Diplôme de Master

Thème:

Simulation et Analyse d'un Réseau Peer-To-Peer Structuré avec DHT Basée sur Chord

Présenté par: *Smari Abderraouf*

Encadrant: *Baaziz Abdelhalim* *Maitre de conférences* *Badji Mokhtar Annaba*

Jury de Soutenance:

Farah Nadir	Professeur	Badji Mokhtar Annaba	Président
Baaziz Abdelhalim	Maitre de conférences B	Badji Mokhtar Annaba	Encadrant
Sari Toufik	Maitre de conférences A	Badji Mokhtar Annaba	Examinateur

2020/2021

REMERCIEMENTS

Je tiens tout d'abord à adresser mes remerciements et ma gratitude à mon superviseur M.BAAZIZ Abdelhalim, pour la qualité de son encadrement, sa patience, sa disponibilité, son aide et son support tout au long de ce projet.

Messieurs Farah Nadir Professeur et Sari Toufik maitre de conférences, à notre université de Badji Mokhtar Annaba d'avoir accepté de faire partie du jury pour examiner ce projet.

J'aimerais remercier ma famille et surtout mes parents d'avoir donné le temps et l'environnement pour réaliser mon projet.

Ainsi que mes frères qui m'ont aidé à le réaliser en me donnant les informations nécessaires pour le rendre possible.

Enfin, que tous celles et ceux qui m'ont apporté leur appui trouvent ici l'expression de mes sincères remerciements.

DEDICACES

A mes chers parents, pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de mes études,

A mes chers frères, pour leur appui et leur encouragement,

A toute ma famille pour leur soutien tout au long de mon parcours universitaire,

Que ce travail soit l'accomplissement de vos vœux tant allégués, et le fuit de votre soutien infaillible,

Merci d'être toujours là pour moi.

RESUME

Les superpositions et les applications pair-à-pair sont très importantes dans les applications quotidiennes actuelles. À l'avenir, cela semble être encore plus pertinent. Les technologies pair-à-pair apportent des avantages en termes de contrôle décentralisé, d'optimisation des ressources et de résilience. Les protocoles pair-à-pair sont généralement conçus pour connecter un très grand nombre de nœuds, donc la simulation est un outil essentiel pour aider à créer des protocoles d'application pair-à-pair et évaluer les caractéristiques du protocole. Nous allons proposer un simulateur basé cycle d'un réseau pair-à-pair structuré avec DHT basée sur Chord avec JAVA.

Mot clés: *pair-à-pair, DHT, Chord, JAVA, Simulateur*

ABSTRACT

Overlays and peer-to-peer applications are very important in today's everyday applications. In the future, this seems to be even more relevant. Peer-to-peer technologies provide advantages in terms of decentralized control, resource optimization and resiliency. Peer-to-peer protocols are typically designed to connect a very large number of nodes so simulation is an essential tool to help create peer-to-peer application protocols and evaluate protocol characteristics. We will propose a cycle-based simulator of a Structured Peer-to-Peer Network with DHT based on Chord with JAVA.

Keywords: *peer-to-peer*, *DHT*, *Chord*, *JAVA*, *simulator*

الملخص

تعد انظمة نظير إلى نظير مهمة جدًا في التطبيقات اليومية ،وسيصبح اكثر اهمية في المستقبل. توفر تقنيات نظير إلى نظير مزايا من حيث التحكم اللامركزي والقيمة مقابل المال والمرونة. عادةً ما يتم تصميم بروتوكولات هذا النوع من التطبيقات لتوصيل عدد كبير جدًا من العقد ، لذا تعد المحاكاة أداة أساسية للمساعدة في إنشائها وتقييم خصائص البروتوكول. سنقترح محاكاة قائمة على دورة لشبكة نظير إلى نظير منظمة مع جداول التجزئة الموزعة DHT على أساس Chord باستخدام لغة البرمجة JAVA.

الكلمات الرئيسية: *نظير إلى نظير , Java , Chord, DHT*

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

LISTE DES TABLEAUX

CHAPITRE 1: INTRODUCTION

Introduction

Aux grands systèmes distribués à l'échelle d'Internet, tels que le World Wide Web, les inconvénients du modèle client-serveur standard deviennent évidents : Les ressources individuelles sont concentrées sur un ou un petit nombre de nœuds et afin de fournir un accès de 24 heures sur 24 et 7 jours sur 7 avec des temps de réponse acceptables, des algorithmes sophistiqués d'équilibrage de la charge et de tolérance aux pannes doivent être appliqués. Il en va de même pour la bande passante du réseau qui vient s'ajouter à ce scénario de goulot d'étranglement. Ces deux problèmes principaux ont incité les chercheurs à proposer des approches permettant de répartir la charge de traitement et la bande passante du réseau entre tous les nœuds participant à un système d'information distribué. Cela résout les problèmes de goulot d'étranglement mais doit être "payé" par une complexité algorithmique considérablement plus élevée et des problèmes de sécurité supplémentaires.

Les systèmes et les applications peer-to-peer sont des systèmes distribués sans contrôle centralisé ou organisation hiérarchique, où les logiciels exécutés sur chaque nœud sont équivalents en termes de fonctionnalité. Un aperçu des caractéristiques des applications peer-to-peer récentes donne une longue liste: stockage redondant, permanence, sélection de serveurs proches, anonymat, recherche, authentification et nommage hiérarchique. En revanche cette richesse de fonctionnalités, l'opération centrale de la plupart des systèmes peer-to-peer est la localisation efficace des éléments de données.

Le protocole Chord ne supporte qu'une seule opération: étant donné une clé, il fait correspondre la clé à un nœud. En fonction de l'application qui utilise Chord, ce nœud peut être responsable du stockage d'une valeur associée à la clé. Chord utilise une variante de hachage cohérent [1] pour affecter les clés aux nœuds Chord. Le hachage cohérent tend à équilibrer la charge, car chaque nœud reçoit à peu près le même nombre de clés, et implique relativement peu de mouvements de clés lorsque les nœuds rejoignent et quittent le système.

Les simulateurs de réseaux modernes sont des outils de première importance lorsque l'on désire obtenir des informations relatives aux performances des réseaux et que toute approche plus théorique ne semble pas possible. Dans la majorité des cas, une analyse réaliste des performances

des réseaux nécessite la prise en compte de protocoles existants, des équipements existants (routeur, commutateurs, etc…) et des modèles de trafic réalistes. Cette approche descriptive des réseaux ne permet pas la résolution analytique car le nombre de variables, souvent dépendantes du temps, qu'il est nécessaire de prendre en compte est trop grand. Il existe de nombreux outils pour simuler les réseaux peer-to-peer, depuis les traditionnels simulateurs bas niveau jusqu'aux simulateurs plus spécifiques développés récemment dans le domaine du P2P. Cependant, la problématique du comportement de l'utilisateur est relativement nouvelle dans le domaine des réseaux. De ce fait, elle ne semble être apparue qu'avec les réseaux P2P.

Problématique

Lorsqu'un développeur crée un protocole peer-to-peer, même s'il est considéré analytiquement comme correct, efficace et évolutif, un environnement de test doit être mis en place pour évaluer les caractéristiques du protocole.

Les protocoles peer-to-peer sont généralement conçus pour connecter un très grand nombre de nœuds. Afin de tester le protocole de manière convaincante, un environnement de simulation est nécessaire. En outre, un déploiement à échelle réduite sur un réseau réel est également nécessaire, comment déployer cette nouvelle technologie à moindre coût pour la tester, et à une échelle suffisamment grande pour obtenir des résultats utiles ?

Motivations

Le monde actuel évolue vers plus de décentralisation. Le meilleur exemple est la crypto-monnaie comme Bitcoin, Litecoin, Ethereum etc... Elle a largement dominé le monde avec son approche peer-to-peer. Pour qu'une telle approche réussisse, la localisation efficace des clés est d'une grande importance et le protocole Chord trouve donc son importance dans le monde actuel.

Les environnements de simulation ont une limite de nœuds qu'ils peuvent simuler. En maintenant la compatibilité des API au prix d'une pénalité de performances.

Solutions

Pour contourner les limites et lourdeurs des simulateurs qui existent pour les réseaux peer-to-peer, nous allons réaliser un outil de simulation et d'analyse d'un réseau Peer-To-Peer structuré avec DHT basée sur Chord.

Esquisse de la solution

Pour réaliser notre solution, nous allons suivre les étapes suivantes:

- Étude du comportement du protocole Chord.
- Étude des différents types de simulateurs.
- Proposer un modèle de simulation dirigée par le cycle de notre simulateur.
- Adaptation du protocole Chord par notre simulateur.
- Déterminer les mesures pour les analyses.

Objectives

Les simulateurs peer-to-peer actuels sont des outils incomplets. Ils contraignent les développeurs en limitant le type et l'étendue des simulations qu'il est possible d'effectuer. Les limitations de mémoire, le manque de simplicité des API, les mauvaises performances sont quelques-uns des problèmes qui affectent le domaine des simulateurs peer-to-peer. Ce mémoire vise à remplir les objectifs suivants:

- Proposer et implémenter une simulation dirigée par le cycle d'une architecture de réseau peer-to-peer basée sur Chord.
- Faire des analyses des résultats mesurées.

Contenu du mémoire

Nous avons eu à faire dans le premier chapitre une introduction générale de ce mémoire, la solution qu'on va proposer et les objectives de ce travail pour vérifier la validité de nos hypothèses et de répondre à la problématique posée, nous adopterons dans ce travail, un plan constitué d'une introduction et trois chapitres:

Le deuxième chapitre sera consacré à la définition des réseaux peer-to-peer et les concepts de base de cette architecture, les avantages et les Inconvénients et la classification des réseaux de ce type puis une comparaison entre les différentes architectures P2P. Ensuite dans la deuxième partie de ce chapitre nous aborderons le protocole Chord qui est un réseau de recouvrement de type table de hachage distribuée pour les réseaux pair à pair et l'explication des déférents algorithmes de ce protocole

Le troisième chapitre sera la simulation des réseaux peer-to-peer et leurs types, nous parlerons aussi des simulateurs existants et leurs avantages et inconvénients. Ensuite nous allons discuter notre solution qui est un simulateur d'un réseau peer-to-peer utilisant le langage JAVA.

Le quatrième chapitre présentera l'implémentation du protocole Chord et expliquera la structure de code avec des exemples de quelques méthodes, et l'adaptation de ce protocole à notre simulateur puis l'exposition des résultats obtenus de notre simulateur avec une présentation graphique.

CHAPITRE 2: LES RESEAUX PEER-TO-PEER ET LE PROTOCOLE CHORD

Dans le monde de l'architecture réseau, vous rencontrerez fréquemment les termes "peer-to-peer" et "client/serveur". Les réseaux peer-to-peer et client/serveur connectent des ordinateurs afin qu'ils puissent partager des ressources d'un ordinateur à un autre, comme les fichiers.

L'architecture client-serveur est l'approche la plus couramment utilisée pour le transfert de données. Elle désigne un ordinateur comme serveur et un autre comme client. Dans une architecture client-serveur, le serveur doit être en ligne en permanence et bénéficier d'une bonne connectivité. Le serveur fournit des données à ses clients et peut également recevoir des données des clients. Quelques exemples d'applications client-serveur largement utilisées sont http, FTP et RSYNC. Toutes ces applications ont une fonctionnalité spécifique côté serveur qui met en œuvre le protocole.

Dans ce chapitre, nous allons décrire les différents concepts des réseaux P2P. Nous donnerons quelques définitions, les motivations pour utiliser ces systèmes et leurs principales caractéristiques. Ensuite, nous commencerons la partie relative aux types d'applications des réseaux P2P. Enfin, nous détaillerons les différentes architectures des systèmes P2P, ainsi que les avantages et les inconvénients de chaque type d'architecture. Ensuite nous décrirons le protocole Chord, en incluant quelques exemples de pseudocode des fonctions. Le protocole contient des fonctions permettant de localiser les nœuds et de traiter les rejointes et les échecs des nœuds.

Concepts de base des réseaux peer-to-peer

1. Définitions

Lors de notre revue de la littérature, nous avons trouvé plusieurs définitions des systèmes peer-to-peer, nous en citons ici deux parmi celles proposées par la communauté:

" *Une architecture de réseau distribué peut être appelée réseau Peer-to-Peer (P-to-P, P2P...), si les participants partagent une partie de leurs propres ressources matérielles (puissance de traitement, capacité de stockage, capacité de liaison réseau, imprimantes...). Ces ressources partagées sont nécessaires pour fournir le service et le contenu offerts par le réseau (par exemple, le partage de fichiers ou les espaces de travail de contenu pour la collaboration). Elles sont accessibles par les autres pairs directement, sans passer par des entités intermédiaires. Les participants d'un tel réseau sont donc aussi bien des fournisseurs de ressources que des demandeurs de ressources (concept de Servent). "* [2]

"Les systèmes peer-to-peer sont des systèmes distribués constitués de nœuds interconnectés capables de s'auto-organiser en topologies de réseau dans le but de partager des ressources telles que le contenu, les cycles d'unité centrale, le stockage et la bande passante, capables de s'adapter aux défaillances et d'accueillir des populations transitoires de nœuds tout en maintenant une connectivité et des performances acceptables, sans nécessiter l'intermédiation ou le soutien d'un serveur ou d'une autorité centralisée globale." [3]

A partir de ces définitions, on peut dire que la technologie P2P permet non seulement le partage de ressources numériques (textes, sons, images, etc.), mais aussi le partage de capacités de traitement de l'information et de communication, espace de stockage (CPU, RAM, etc.), ainsi l'une des caractéristiques les plus marquantes des réseaux P2P est que les échanges se font directement entre les utilisateurs qui peuvent être à la fois contributeurs et consommateurs.

2. Caractéristiques des réseaux peer-to-peer

Dans cette section, nous présentons les principales caractéristiques du modèle peer-to-peer [4]:

- **Décentralisation**: le fait que chaque nœud gère ses propres ressources évite la centralisation du contrôle. Un système P2P peut fonctionner sans avoir besoin d'une administration centralisée, ce qui permet d'éviter les goulets d'étranglement et d'augmenter la résilience du système aux pannes et aux défaillances.
- **Mise à l'échelle**: Il s'agit d'amener un grand nombre de nœuds (jusqu'à des milliers ou des millions) à coopérer pour partager leurs ressources tout en maintenant de bonnes performances du système. Cela signifie qu'un système P2P doit offrir des méthodes bien adaptées à un environnement dans lequel il y a un grand volume de données à partager, un grand nombre de messages à échanger entre un grand nombre de nœuds partageant leurs ressources via un réseau largement distribué.
- **L'auto-organisation**: Les systèmes P2P étant souvent déployés sur Internet, la participation d'un nouveau nœud à un système P2P ne nécessite pas une infrastructure coûteuse. Il suffit

d'avoir un point d'accès à Internet et de connaître un autre nœud déjà connecté pour se connecter au système. Un système P2P doit être un environnement ouvert, c'est-à-dire qu'un utilisateur sur un nœud doit pouvoir connecter son nœud au système sans avoir besoin de contacter une personne et sans avoir besoin de passer par une autorité centrale.

- **Connectivité ad hoc**: La nature ad hoc de la connectivité a un effet important sur toutes les classes de systèmes P2P. Dans le calcul distribué, les applications parallélisées ne peuvent pas être exécutées sur tous les systèmes tout le temps ; certains des systèmes seront disponibles tout le temps, d'autres seront disponibles une partie du temps et d'autres encore ne seront pas disponibles du tout. Les systèmes P2P et les applications de calcul distribué doivent être conscients de cette nature ad hoc et être capables de gérer les systèmes qui rejoignent et quittent le pool de systèmes P2P disponibles. Alors que dans les systèmes distribués traditionnels, il s'agissait d'un événement exceptionnel, dans les systèmes P2P, il est considéré comme habituel.
- **Autonomie des nœuds**: chaque nœud gère ses ressources de manière indépendante. Il décide de la partie de ses données à partager. Il peut se connecter ou / et se déconnecter à tout moment. Il a également l'autonomie de gérer sa puissance de calcul et sa capacité de stockage.
- **Hétérogénéité**: En raison de l'autonomie des nœuds aux architectures matérielles et/ou logicielles hétérogènes, les systèmes P2P doivent disposer de techniques adaptées pour résoudre les problèmes liés à l'hétérogénéité des ressources.
- **Dynamique**: en raison de l'autonomie des nœuds, chaque nœud peut quitter le système à tout moment, ce qui élimine ses ressources du système. De nouvelles ressources peuvent être ajoutées au système lors de la connexion de nouveaux nœuds. Ainsi, en raison de l'instabilité des nœuds, les systèmes P2P doivent être capables de gérer un grand nombre de ressources très variables. La sortie d'un nœud du système (ou la défaillance d'un nœud) ne doit pas faire échouer le système. Elle doit être tolérée et avoir un "petit" impact sur les performances de l'ensemble du système.
- **Sécurité**: Les systèmes P2P partagent la plupart de leurs besoins de sécurité avec les systèmes distribués courants: chaînes de confiance entre pairs et objets partagés, schémas d'échange de clés de session, cryptage, digests numériques et signatures. Des recherches approfondies ont été menées dans ces domaines, et nous n'en parlerons pas davantage dans le présent document. De nouvelles exigences de sécurité sont apparues avec les systèmes P2P.

3. Objectifs des réseaux peer-to-peer

L'objectif principal des systèmes P2P est de répondre aux besoins des utilisateurs, comme tout système informatique. Les réseaux P2P sont apparus pour résoudre certains problèmes rencontrés dans le paradigme client/serveur. Les atouts pour lesquels les internautes préfèrent l'utilisation des systèmes P2P sont souvent les critères suivants. [5]

- **Coût**: plusieurs processeurs à faible coût.
- **Puissance de calcul**: exploite les ressources de l'ordinateur.
- **Performance**: calcul parallèle.
- **Fiabilité**: résistance aux défaillances logicielles ou matérielles.
- **Extensibilité**: croissance progressive en fonction des besoins.

4. Avantages et inconvénients des réseaux P2P

Selon les propriétés citées ci-dessus, nous pouvons affirmer que les systèmes P2P sont plus adaptés aux environnements qui ont de grandes quantités de ressources à partager.

4.1. Avantages

- Éviter la création de goulots d'étranglement.
- Utiliser des ressources importantes réparties sur un grand nombre de nœuds du réseau.
- Améliorer l'utilisation de la bande passante, des ressources de traitement et des espaces de stockage.
- Accélérer l'accomplissement des tâches en réduisant le temps de traitement grâce à des liens directs entre pairs.
- Éviter le point de défaillance unique par la distribution redondante des ressources et la décentralisation des systèmes P2P. Cette caractéristique offre à ce type de système plus de fiabilité et de robustesse.
- Augmenter les performances du système en partageant la charge de travail et en augmentant l'autonomie et l'agrégation des ressources, ce qui accroît les performances des réseaux P2P.
- Permettre aux utilisateurs de garder le contrôle de leurs ressources. En outre, ils peuvent rejoindre ou quitter le système à tout moment.
- Flexibilité (politiques d'investissement).

4.2. Inconvénients

Bien que les réseaux P2P soient très populaires et très utiles aujourd'hui, ils présentent des inconvénients, nous en citons quelques-uns [6]:

- **Problèmes de comportement des utilisateurs**: Les systèmes de partage de fichiers peer-to-peer peuvent facilement souffrir d'une banalité générale. En raison de l'anonymat, les membres sont parfois tentés d'adopter un comportement malveillant. Sans un minimum de contrôle, des activités telles que la diffusion de virus ou le parasitisme peuvent se produire.
- **Les problèmes de comportement des entreprises**:
    - Une atteinte à la vie privée: Notre vie privée n'est pas forcément protégée lors de l'utilisation d'outils peer-to-peer. En effet, les adresses IP des utilisateurs peuvent être récupérées lorsque le logiciel est centralisé.
    - La pollution des réseaux: Sans même annoncer d'ultimatum, plusieurs sociétés proposent de polluer les réseaux d'échange de musique entre particuliers en y intégrant des fichiers de mauvaise qualité ou incorrects. L'objectif est de rendre moins attractifs ces réseaux libres et anarchiques. Un ensemble de sociétés indépendantes développent de nouvelles technologies pour polluer les réseaux. Leurs clients potentiels sont les industries de la musique et du cinéma, qui cherchent par tous les moyens à détourner les utilisateurs de téléchargements gratuits vers des systèmes sécurisés et payants.
- **La propagande**: Il est bien connu que les systèmes peer-to-peer sont le plus souvent investis par des bandes publicitaires, même si aujourd'hui les versions légères des logiciels sont proposées sans publicité.

Classification de l'architecture peer-to-peer

Comme le montre la figure 1, les systèmes informatiques peuvent être classés en deux catégories différentes: les systèmes centralisés et les systèmes distribués, ces derniers étant à leur tour divisés en deux: les modèles client/serveur et les modèles pair-à-pair. Les premiers peuvent être soit pur dans le cas où les clients n'interagissent qu'avec un seul serveur, soit hiérarchiques dans le cas où les clients n'ont de contact qu'avec les serveurs de plus haut niveau.

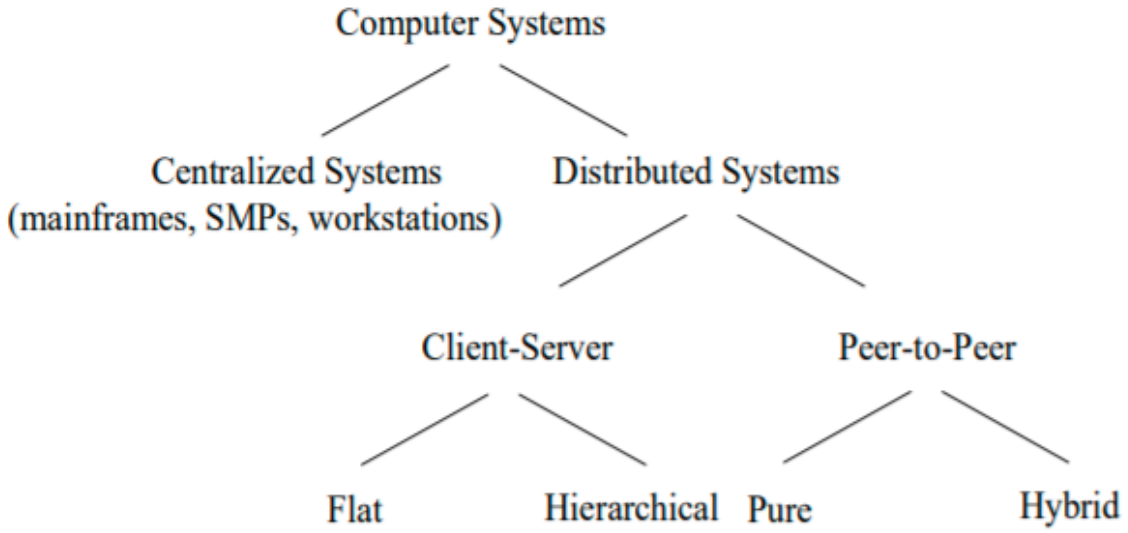


Figure 1: Une taxonomie des architectures de systèmes informatiques

Le modèle peer-to-peer peut être centralisé, hybride ou pur, en fonction de la manière dont le contenu est recherché.

1. Peer-to-peer décentralisé (pur)

Dans ce type d'architecture, il n'y a plus de serveur central, tous les nœuds sont égaux et jouent le même rôle, ainsi la suppression d'un pair du réseau n'affecte pas les services offerts (voir Figure 2), par exemple en revanche, l'absence d'un serveur central ayant une vue globale sur la localisation des ressources hébergées par les pairs dans le réseau P2P pose le problème suivant: comment un pair peut-il découvrir et accéder à une ressource dans ce contexte ? Pour cela, deux solutions ont été proposées: la première basée sur le flooding et la seconde sur les tables de hachage distribuées "DHT" [7], également connues respectivement sous le nom de réseaux P2P décentralisés non structurés et de réseaux P2P décentralisés structurés.

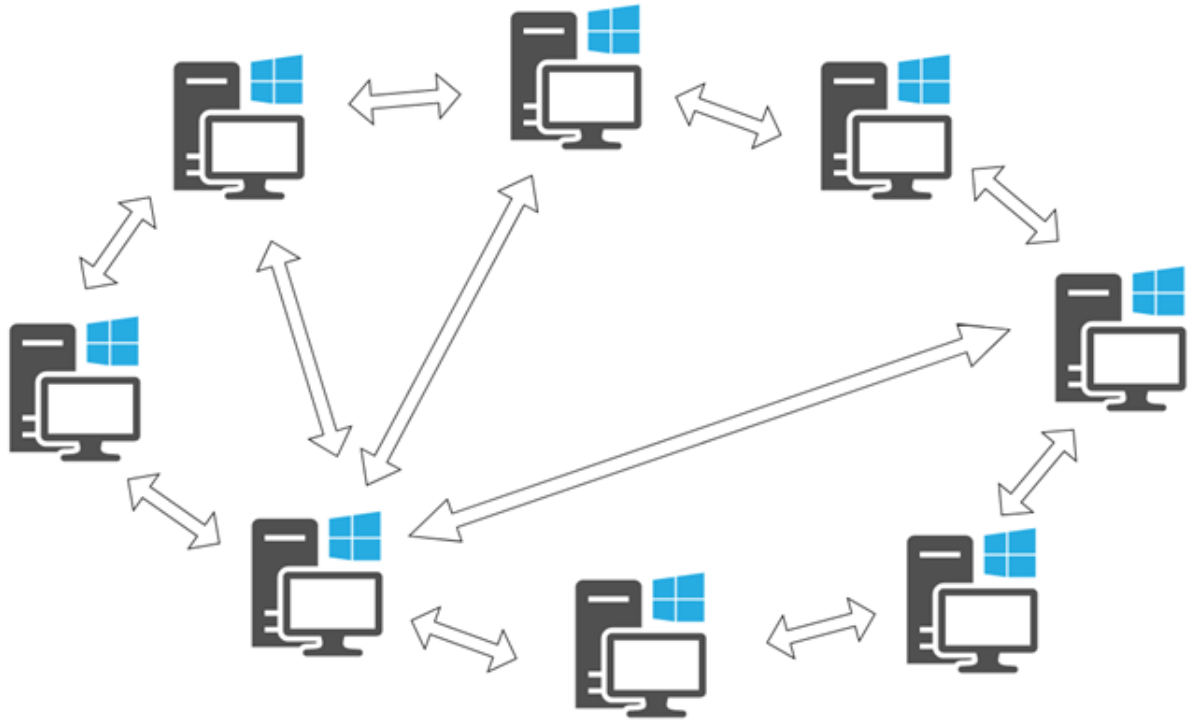


Figure 2: réseau décentralisé de pair à pair

1.1. Architecture non structurée

Historiquement, ce type d'architecture représente la deuxième génération de réseaux P2P. Une architecture non structurée n'a pas de topologie spécifique. Chaque nœud a une vue limitée du réseau, il n'est connecté qu'à 3 ou 4 nœuds au maximum, et connaît tous les pairs qui sont à 7 sauts maximum. Lorsqu'un nouvel utilisateur veut rejoindre le réseau: il envoie un message de diffusion pour savoir quels autres nœuds du réseau sont connectés [8].

Pour découvrir une ressource, une requête sera transmise d'un pair à l'autre jusqu'à atteindre le client qui possède l'objet désiré, afin d'éviter d'inonder le réseau pendant un temps trop long, le système associe à chaque requête un compteur TTL "Time to Live", la valeur attribuée au TTL est généralement de 7. Lorsqu'il atteint zéro, la demande n'est plus renvoyée.

L'inconvénient majeur de ce mécanisme vient de l'expiration du TTL avant que l'ensemble du réseau ait été recherché, ce qui peut conduire à un échec de la recherche même si l'objet désiré est disponible sur le réseau P2P, il n'est pas bien dimensionné et génère une surcharge du réseau par les trames diffusées. Cette méthode est utilisée dans le système Gnutella [GNUTELLA]

1.2. Architecture Structurée

Pour résoudre les problèmes posés dans les réseaux non structurés, un autre type d'architectures purement décentralisées est apparu. Ce sont des systèmes P2P structurés qu'une table de hachage distribuée "DHT" est une structure de données permettant la découverte et la localisation efficace de ressources via des clés, chaque pair est responsable d'une partie de la table de hachage, cette dernière utilise une fonction de hachage sécurisée telle que SHA-1 [9] et MD5, Le hachage de l'adresse IP donne l'identifiant d'un utilisateur tandis que celui du nom d'une ressource donne la clé. Il s'agit alors de stocker de manière distribuée les couples (clé, identifiant) sur les nœuds du réseau afin que chaque ressource du réseau soit associée à l'adresse de l'utilisateur propriétaire de la ressource. Une redondance dans le stockage est également introduite afin que la sortie d'un nœud du système ne perde pas les métadonnées qu'il stocke et ne rende pas impossible l'accès à ces données. La fonction de hachage utilisée garantit que pour deux ressources différentes, les clés générées seront identiques.

Les propriétés suivantes sont conférées aux DHT:

- **La fiabilité:** L'utilisation d'un algorithme de découverte et de routage permet, pour une clé donnée, de déterminer le pair identificateur le plus proche. Dans des conditions statiques, une réponse négative à une requête signifie que la ressource requise n'est pas disponible dans la communauté.
- **La performance:** Dans des conditions normales de fonctionnement, le nombre de sauts nécessaires est limité. Par exemple, dans une communauté de 106 pairs utilisant un identifiant hexadécimal, la longueur moyenne des requêtes est d'environ cinq sauts.
- **La mise à l'échelle:** Deux caractéristiques confèrent aux DHT de bonnes performances de mise à l'échelle. La première est liée au nombre moyen de sauts nécessaires pour router les requêtes, qui reste faible, même dans le cas de communautés avec un grand nombre de participants. La seconde concerne les tables de routage, qui restent également d'une taille raisonnable par rapport au nombre de participants.
- **La tolérance aux pannes:** En raison de l'absence de centralisation, qui exclut tout point central, les DHT présentent une bonne tolérance aux suppressions aléatoires de nœuds. Les requêtes peuvent être acheminées même si certains des nœuds disparaissent. En revanche, chaque nœud racine d'une ressource particulière ressemble à un point central. Des mécanismes de redondance sont souvent mis en place pour éviter l'inaccessibilité d'une ressource présente dans une DHT. Exemples de DHT actuelles: Pastry, Chord, CAN ... Nous verrons plus en détail Chord dans la deuxième partie de ce chapitre.

2. Architectures semi-centralisées (hybrides)

Dans la pratique, les nœuds d'un réseau P2P ont des capacités de stockage et de traitement caractérisées par une forte disparité qui approuve la création du modèle "hybride" qui est un mélange entre le mode P2P et l'architecture client/serveur.

Les réseaux P2P hybrides utilisent un nombre suffisant de serveurs (appelés super-pairs) pour éviter le risque de disparition de l'un d'entre eux. Chaque serveur joue le rôle de nœud central d'un groupe (cluster) qui comprend un ensemble de pairs organisés en P2P. Les super-pairs peuvent également jouer le rôle de clients. La figure suivante présente l'architecture hybride:

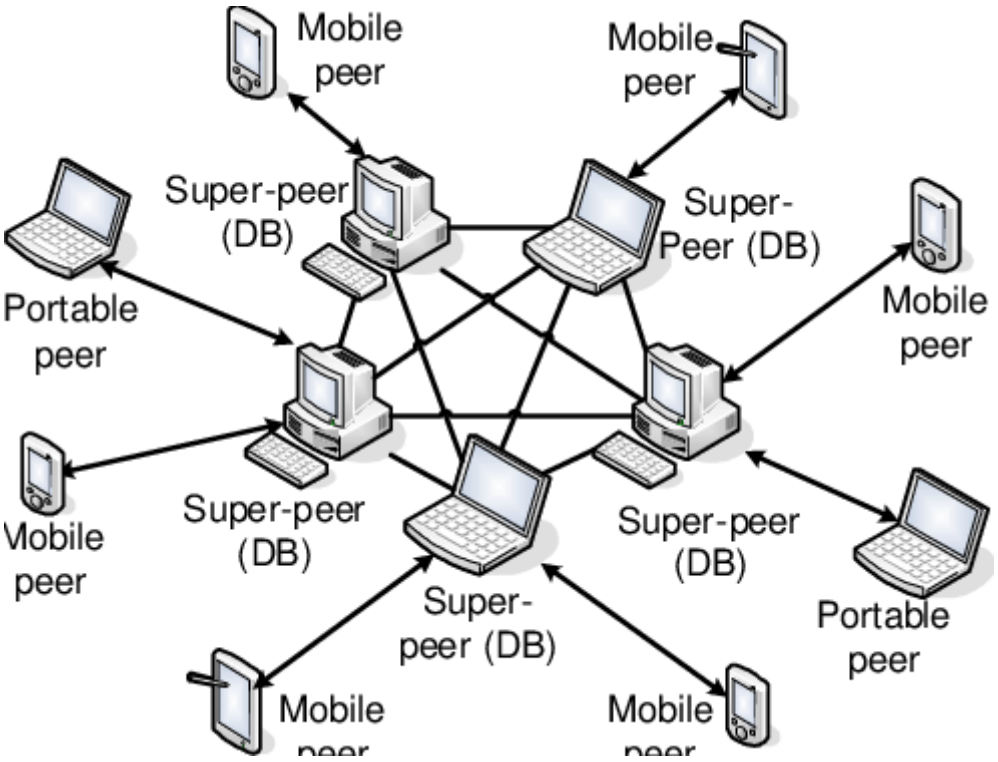


Figure 3: Architecture hybride

Il existe deux types de réseaux P2P hybrides:

- **P2P hybride statique:** les super-pairs sont sélectionnés manuellement dès la création du réseau. Cette approche présente fondamentalement les mêmes défauts que les systèmes centralisés.
- **P2P hybride dynamique:** le réseau peut changer ses super-pairs si les bonnes conditions sont réunies. Le système peut décider si un nœud est promu au rang de super-pair ou si un super-pair est rétrogradé au rang de nœud normal, selon le protocole et les critères utilisés.

Les réseaux P2P hybrides présentent certains des avantages de la centralisation et de la décentralisation, mais aussi certains de leurs inconvénients. La tolérance aux pannes est une préoccupation majeure dans ces systèmes, surtout dans ceux où les super-pairs sont statiques. De plus, la gestion des clusters est une tâche délicate, surtout lorsqu'un nœud peut prendre part à plusieurs groupes en même temps.

La comparaison entre les différentes architectures P2P est donnée dans le tableau 1:

Architecture	**Centralisé**	**Décentralisé**	**Semi-centralisé**
Évolution du réseau	Moyen	Faible	Très élevé
Résilience	Faible	Très élevé	Moyen
Efficacité de la recherche	Très élevé	Moyen	Haut
Couverture de recherche	Très élevé	Moyen	Haut
Couverture de recherche	Très élevé	Faible	Haut
Surcharge de signalisation dans la PS	Très élevé	__	Haut
Surcharge de signalisation dans les OP	Faible	Haut	Faible
Protocoles utilisés	Dépend du système	Inondations/DHT	Dépend du système

Tableau 1: Comparaison de différentes architectures P2P.

Applications et services qui utilisent peer-to-peer

Dans cette section, nous donnons un bref aperçu des principales applications et de leur émergence en tant que services grand public pour des millions d'utilisateurs. [9]

1. Applications peer-to-peer de partage de fichiers

Près de dix ans après l'apparition du World Wide Web sur Internet, les applications décentralisées de partage de fichiers de poste à poste ont supplanté l'application Napster [NAPSTER], basée sur un serveur, qui avait popularisé le concept de partage de fichiers. Les répertoires centralisés de Napster constituaient son talon d'Achille car, comme il a été affirmé devant les tribunaux, Napster avait les moyens, par le biais de ses serveurs, de détecter et d'empêcher l'enregistrement de contenus protégés par le droit d'auteur dans son service, mais il ne l'a pas fait. Napster a ensuite été jugé responsable de la violation des droits d'auteur, ce qui a porté un coup fatal à son modèle commercial.

Alors que Napster était rongé par les poursuites judiciaires, les protocoles de deuxième génération tels que Gnutella [GNUTELLA], FastTrack [FASTTRACK] et BitTorrent [BITTORRENT] ont adopté une architecture pair-à-pair dans laquelle il n'y a pas de répertoire central et où toutes les recherches et tous les transferts de fichiers sont répartis entre les pairs correspondants. D'autres systèmes, comme FreeNet, ont également intégré des mécanismes d'anonymat des clients, notamment en acheminant les demandes indirectement par d'autres clients et en cryptant les messages entre pairs. Pendant ce temps, les principaux labels de l'industrie de la musique, qui ont sans doute subi les pertes de revenus les plus importantes en raison de l'émergence du partage de fichiers, ont continué à contester juridiquement ces systèmes et leurs utilisateurs.

Quelle que soit l'issue de ces affaires judiciaires, la perception sociale de l'acceptabilité et des avantages de la distribution de contenu par le biais d'applications P2P a été irrévocablement modifiée. Dans l'industrie de la musique, avant le partage de fichiers P2P, les CD audio étaient le mécanisme de distribution dominant. Les portails Web de musique en ligne étaient limités en termes de taille de leurs catalogues et les téléchargements étaient coûteux. Bien que le partage de fichiers P2P ait été largement assimilé au piratage de contenu, il a également montré que les consommateurs étaient prêts à remplacer le modèle de distribution des CD par une expérience en ligne si celle-ci pouvait offrir un large portefeuille de titres et d'artistes et si elle comportait des

fonctionnalités telles que la recherche, les prévisualisations, le transfert vers des CD et des lecteurs de musique personnels, et l'achat de titres individuels. L'apparition de portails tels qu'iTunes [ITUNES] offrant ces propriétés a entraîné une croissance considérable du secteur de la musique en ligne.

1.1 La voix sur P2P (VoP2P)

Skype [SKYPE] est un client VoP2P lancé en 2003 qui a atteint plus de 10 millions d'utilisateurs simultanés. Par rapport aux clients VoIP antérieurs, Skype propose à la fois des appels gratuits de bureau à bureau et des appels à faible coût de bureau à réseau téléphonique public commuté (RTPC), y compris des appels internationaux. La qualité des appels est élevée, ce qui est généralement attribué au codec audio utilisé par Skype et à la large utilisation actuelle des réseaux d'accès à large bande pour accéder à Internet. En outre, Skype inclut des fonctions des applications IMP, notamment les listes d'amis, la messagerie instantanée et la présence. Contrairement aux systèmes de partage de fichiers, Skype promet une politique sans logiciel espion [10].

1.2 P2PTV

Le succès du partage de fichiers P2P et de la VoP2P a motivé l'utilisation du P2P pour les applications de diffusion vidéo en continu. La diffusion P2PTV suit souvent une organisation en canaux dans laquelle le contenu est organisé et accessible selon un répertoire de programmes et de films. Contrairement aux systèmes de partage de fichiers dans lesquels un fichier multimédia est d'abord téléchargé sur l'ordinateur de l'utilisateur puis lu localement, les applications de streaming vidéo doivent fournir à chaque pair un taux de transfert de flux en temps réel égal au taux de lecture vidéo. Ainsi, si un flux multimédia est codé à 1,5 Mbps et qu'un seul homologue est la source du flux, le chemin entre l'homologue source et l'homologue de lecture doit fournir un taux de transfert de données de 1,5 Mbps en moyenne. Une certaine variation du débit de lecture le long du chemin peut être prise en compte en pré-tamponnant un nombre suffisant d'images vidéo. Si le taux de transfert chute temporairement, le contenu supplémentaire de la mémoire tampon est utilisé pour éviter les interruptions de service du côté du rendu [10].

Plusieurs applications P2PTV sont disponibles. Par exemple, Babelgum, Joost, PPLive, PPStream, SopCast, TVants, TVUPlayer, Veoh TV et Zattoo.

2. Le calcul en grille

L'un des premiers moteurs de l'informatique P2P a été la prise de conscience du fait qu'un PC domestique typique est le plus souvent inactif et qu'il pourrait donc être exploité pour résoudre des problèmes de calcul complexes. L'idée est de décomposer le problème en grandes parties qui peuvent être exécutées simultanément avec très peu d'interaction, ce que certains appellent le calcul (parallèle embarrassant) [11] Les exemples les plus connus mais aussi les plus simples de cette utilisation du "partage des cycles de CPU" sont SETI@HOME [SETI] et ses variantes entropia [Entropia], ud [UD]. Une comparaison imagée entre les serveurs de calcul traditionnels et l'approche de SETI@HOME est présentée dans la figure ci-dessous. Récemment, le modèle SETi@HOME a été utilisé avec succès par l'équipe de l'Université d'Oxford pour réduire le nombre de composés possibles pour le traitement de l'anthrax de 3,5 milliards à un nombre plus gérable de 300 000. L'ensemble des données a été recherché en une période relativement courte (4 semaines) au lieu de plusieurs années. Des idées similaires ont été testées pour d'autres problèmes complexes tels que l'analyse par éléments finis, les modèles de changement climatique, le rendu graphique de scènes complexes, etc. Les exemples impliquant le partage d'autres ressources matérielles (par exemple, la mémoire ou le disque) sont également nombreux. Par exemple, l'extension de pair à pair du service populaire i-drive [I-Drive] offre la possibilité de stocker d'énormes quantités de données (peut-être de manière répétée) sans investissement correspondant en matériel de stockage.

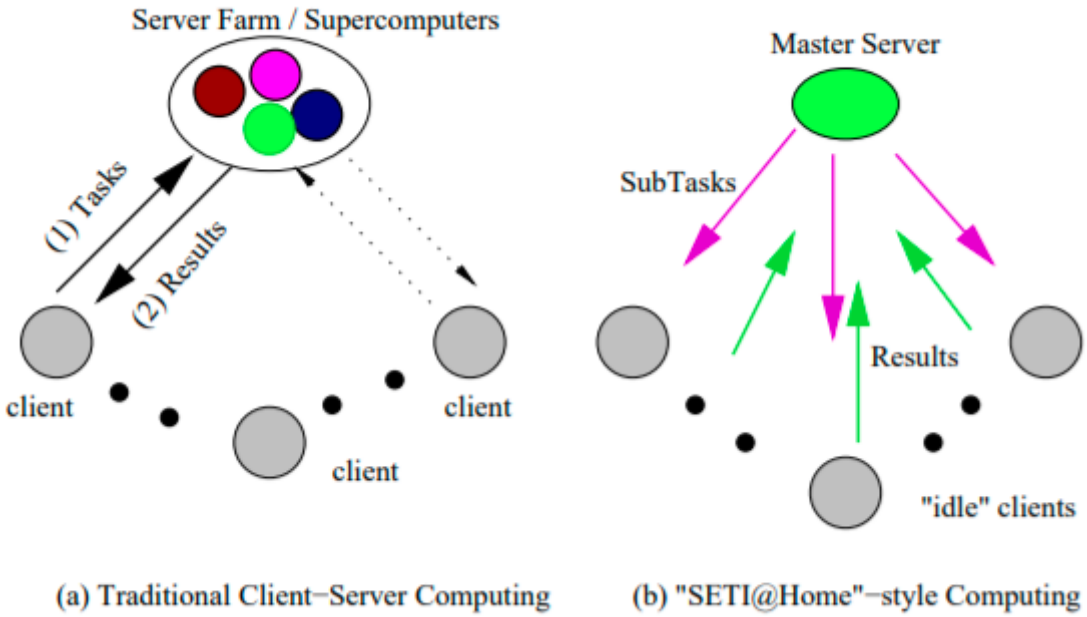


Figure 4: Approche de partage des ressources

3. Moteurs de recherche

L'objectif est d'éviter les limites des moteurs de recherche centralisés qui n'explorent qu'une petite partie du contenu du Web. En effet, le principal problème auquel sont confrontés les moteurs de recherche vient du fait que de plus en plus de pages web sont générées dynamiquement et que les informations qu'elles contiennent proviennent de bases de données auxquelles les moteurs n'ont pas accès. [8]

Les moteurs de recherche P2P sont basés sur une architecture entièrement distribuée. Nous pouvons citer InfraSearch [INFS] [8] comme un moteur de recherche P2P. Ce type de moteur n'exécute pas lui-même la recherche et la récupération d'informations. Il se contente de propager les requêtes formulées par l'utilisateur aux systèmes qui lui sont connectés, ces systèmes les transmettant eux-mêmes à leurs voisins sur le réseau. Chaque hôte recevant la requête est entièrement responsable de l'exécution de la recherche sur son propre système et renvoie ensuite les informations jugées pertinentes à InfraSearch. [8]

4. Calculs et communications collaboratifs

L'informatique collaborative désigne un modèle d'utilisation de l'informatique distribuée qui permet principalement aux utilisateurs de travailler ensemble pour trouver une solution à un problème. La dispersion géographique des organisations a conduit à des solutions logicielles qui permettent aux gens de collaborer ensemble. Socialement, la collaboration est une activité de personne à personne et le modèle informatique P2P imite étroitement cette activité. Groove [Groove] est l'une de ces plateformes de collaboration décentralisées qui permet l'interaction de petits groupes. Le cadre de communication de Groove est essentiellement P2P avec intervention du serveur dans le cas de discontinuités du réseau comme les pare-feu, les proxies et les NAT. En outre, le besoin de communiquer entre personnes a également conduit à des avancées significatives dans la messagerie instantanée, qui est une application P2P par excellence.

5. Plateformes de développement

La plupart des logiciels P2P ont été développés de manière spécifique sans référence à des standards propres au p2p, dans le but de standardiser les réseaux P2P, des plateformes sont mises en place pour servir de base au développement d'applications P2P, elles fournissent les fonctionnalités de base: gestion des pairs, attribution d'identifiants, découverte de ressources, communication entre pairs, sécurité. On peut citer la plateforme JXTA développée par ‘Sun Microsystems’.

Le protocole Chord

1. L’anneau du Chord

Le protocole Chord utilise SHA-1 [9] comme fonction de hachage cohérente pour attribuer un identifiant de m bits à chaque nœud et à chaque clé. Les fonctions de hachage cohérentes sont des fonctions de hachage dotées de certaines propriétés avantageuses supplémentaires, c'est-à-dire qu'elles permettent aux nœuds de rejoindre et de quitter le système avec un minimum de perturbations [13]. Le m est un nombre entier qui doit être choisi suffisamment grand pour rendre négligeable la probabilité que deux nœuds ou deux clés reçoivent le même identifiant. La fonction de hachage calcule l'identifiant de la clé en hachant la clé, et l'identifiant du nœud en hachant l'adresse IP du nœud.

La clé et les identifiants des nœuds sont disposés sur un cercle d'identifiants de taille 2^m appelé Chord ring. Les identificateurs sur l'anneau Chord sont numérotés de 0 à 2^{m-1}. Une clé est attribuée à un nœud dont l'identifiant est égal ou supérieur à l'identifiant de la clé. Ce nœud est appelé le nœud successeur de k, désigné par $successor(k)$, et est le premier nœud dans le sens des aiguilles d'une montre à partir de k sur le cercle.

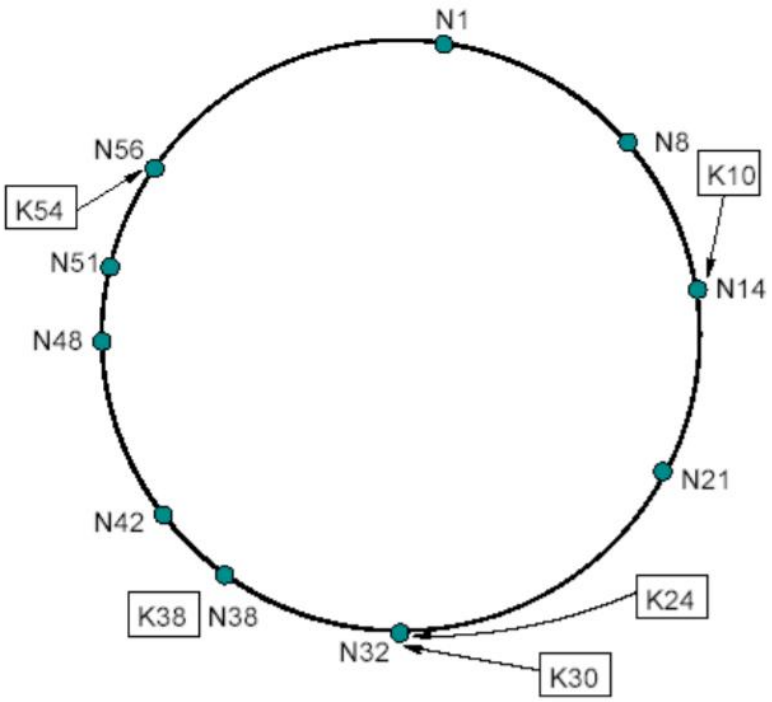


Figure 5: Anneau d'accords avec m = 6, 10 nœuds et 5 clés.

2. Emplacement de la clé

La fonction centrale du protocole Chord est la fonction de localisation des clés. Pour une meilleure compréhension, une fonction simple de localisation de clé est d'abord présentée. Ensuite, la fonction de localisation de la clé évolutive sera démontrée.

2.1 Localisation simple des clés

Pour permettre à un nœud n de trouver une certaine clé k, nous appelons $n.lookup(k)$. Pour exécuter la recherche, le protocole appelle la fonction $find\_successor$ (figure 6), qui renvoie le nœud successeur du nœud n si k se trouve entre n et son successeur ou transmet la requête autour du cercle dans le cas contraire (figure 7).

```
1 // ask node n to find the successor of id
2 n.find successor(id)
3 if (id ∈ (n,successor])
4
5     return successor;
6 else
7 // forward the query around the circle
8
      return successor.find successor(id);
```

Figure 6: Pseudocode for simple key location

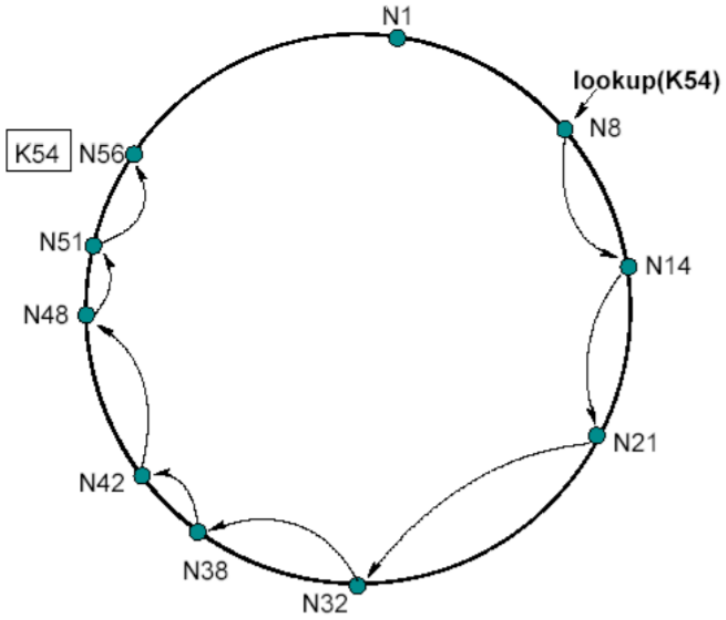


Figure 7: Dans la fonction de recherche simple, la requête est transmise autour du cercle.

Dans le pire des cas, la requête doit être transmise N fois dans un cercle comportant N nœuds, de sorte que le coût d'une consultation est linéaire par rapport au nombre de nœuds. Dans les systèmes comportant un grand nombre de nœuds, les recherches seraient trop lentes. C'est pourquoi Chord utilise une fonction évolutive de localisation des clés qui permet des recherches plus efficaces.

2.2 Localisation évolutive des clés

Afin de fournir des recherches plus efficaces, des informations de routage supplémentaires sont stockées pour accélérer les recherches. Chaque nœud n maintient une table de routage avec jusqu'à m entrées (où m est le nombre de bits des identifiants) qui est appelée la table de hachage. La $i^{ème}$ entrée de la table au niveau des nœuds n contient le premier nœud qui succède à n d'au moins 2^{i-1} , ce nœud s est appelé le $i^{ème}$ entrée des nœuds n. (La figure 8) montre la table de routage du nœud $N8$.

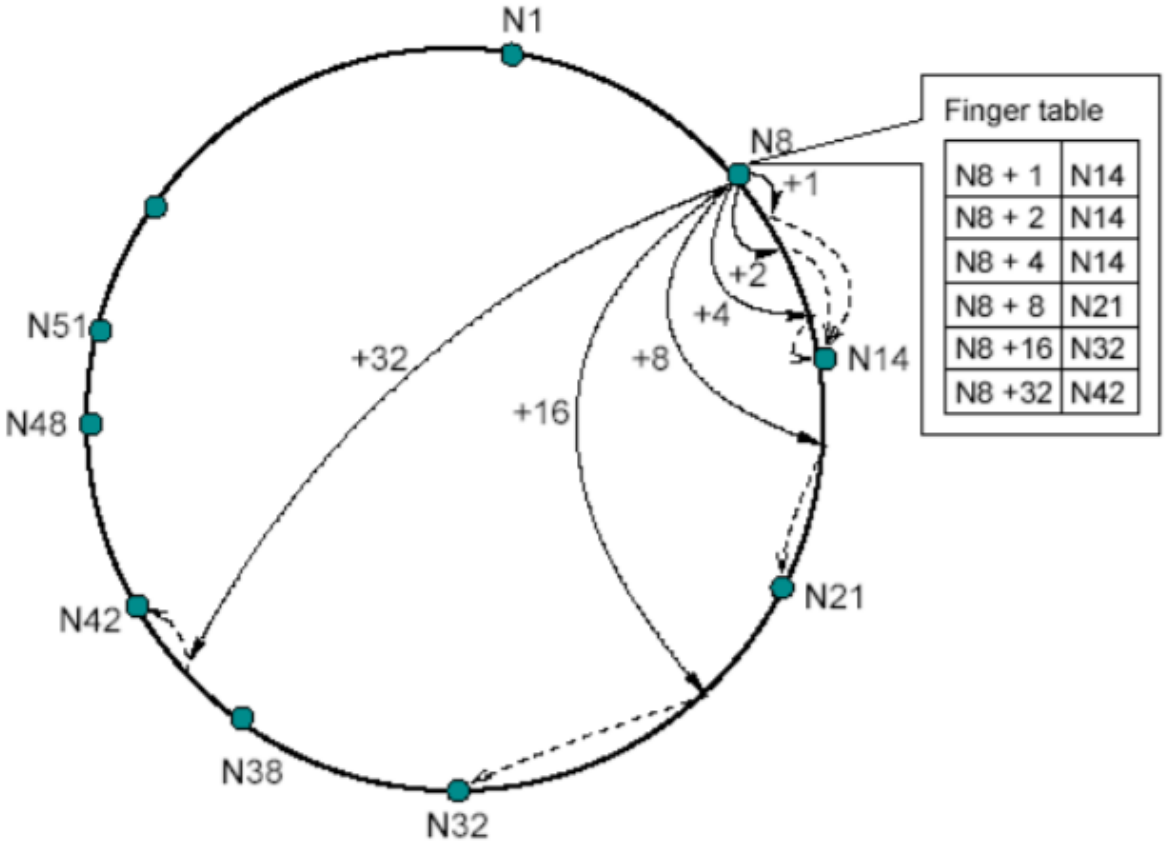


Figure 8 Les entrées du tableau des hachages sont calculées par la formule finger[i] = successor (n + 2^{i-1})

Les caractéristiques importantes de ce système sont les suivantes:

- Chaque nœud ne stocke des informations que sur un petit nombre de nœuds m.
- Chaque nœud en sait plus sur les nœuds qui le suivent de près que sur les nœuds plus éloignés.
- Une table de hachage ne contient généralement pas assez d'informations pour déterminer directement le successeur d'une clé arbitraire k. Un nœud doit contacter d'autres nœuds afin de résoudre la table de hachage.

Lorsqu'on demande à un nœud de trouver une certaine clé, il détermine le prédécesseur le plus élevé de cette clé dans sa table de routage et transmet la clé à ce nœud. Cette procédure déterminera récursivement le nœud responsable de la clé. Le temps de recherche est $O(log\ N)$, puisque la requête est transmise au moins à la moitié de la distance restante autour du cercle à chaque étape (voir Figure 9, Figure 10).

```
1 // ask node n to find the successor of id
2 n.find successor(id)
3 if (id ∈ (n,successor])
4
5     return successor;
6 else
7
8     n0 = closest_preceding_node(id);
9     return n0.find_successor(id);
10 // search the local table for the highest predecessor of id
11 n.closest preceding node(id)
12 for i = m downto 1
13
14     if (finger[i] ∈ (n,id))
15         return finger[i];
16 return n;
```

Figure 9: Pseudocode de la fonction de clé de recherche évolutive

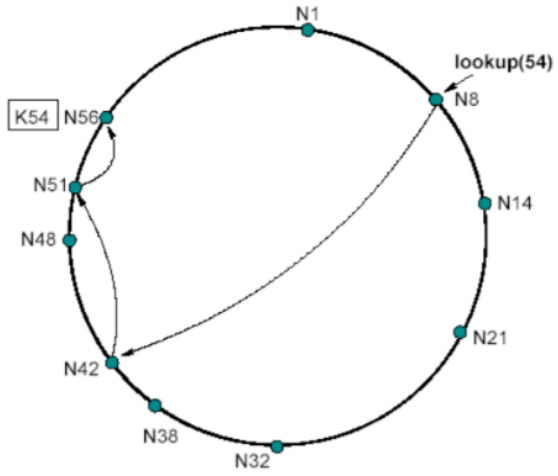


Figure 10: Illustration de la fonction de clé de recherche évolutive

Arrivée de nœuds

Lorsqu'un nœud rejoint le système, les pointeurs de successeur de certains nœuds doivent être modifiés. Il est important que les pointeurs de successeur soient à jour à tout moment car l'exactitude des recherches n'est pas garantie autrement. Le protocole Chord utilise un protocole de stabilisation s'exécutant périodiquement en arrière-plan pour mettre à jour les pointeurs de successeur et les entrées dans la table de hachage. Le pseudocode suivant (Figure 11) explique en détail le fonctionnement du protocole de stabilisation. (Figure 12, Figure 13).

```
// create a new Chord ring.
n.create()

    predecessor = nil;
    successor = n;
// join a Chord ring containing node n0.
n.join(n0)

    predecessor = nil;
    successor = n0.find_successor(n);
// called periodically. verifies n's immediate
// successor, and tells the successor about n.
n.stabilize()

    x = successor.predecessor;
    if (x ∈ (n,successor))
        successor = x;
successor.notify(n);
// n0 thinks it might be our predecessor n.notify(n0)

if (predecessor is nil or n0 ∈ (predecessor, n))

    predecessor = n0;
// called periodically. refreshes finger table entries.

// next stores the index of the next finger to fix.

n.fix_fingers()

    next = next + 1 ;
    if (next > m)
        next = 1 ;
    finger[next] = find successor(n + 2^(next-1));
// called periodically. checks whether predecessor has failed.
n.check predecessor()

    if (predecessor_has_failed)
    predecessor = nil;
```

Figure 11: pseudocode expliquant le fonctionnement du protocole de stabilisation

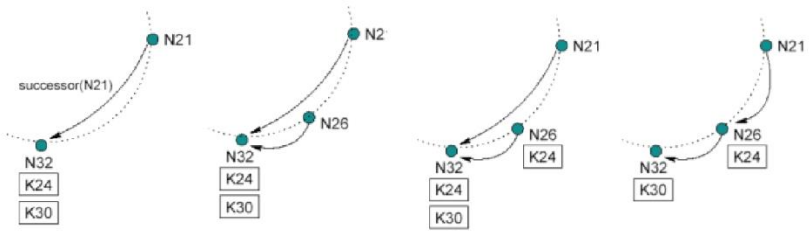


Figure 12: l'illustration de N26 rejoint l'anneau d'accord et N26 copie K24

1. Impact des rejointes de nœuds sur la correction

Lorsque des nœuds se sont rejoints récemment et qu'une recherche se produit avant que la stabilisation ne soit terminée, le système se retrouve dans l'un de ces trois états:

- **Toutes les entrées de la table de hachage et les pointeurs des successeurs sont corrects au moment où la table de hachage est ouverte:** de la recherche: Aucun impact sur l'exactitude, la recherche sera réussie dans le temps. $O(log\ N)$.
- **Les pointeurs de successeur sont corrects, mais les entrées de la table de hachage ne le sont pas**: La recherche sera toujours correcte, mais pourrait être un peu plus lente si un grand nombre de nœuds se sont rejoints entre la cible et le prédécesseur de la cible, la fonction $find\_successor$ sera initialement sous-dépassée et certains des sauts se feront en temps linéaire (figure 14).
- **Ni les entrées de la table de hachage ni les pointeurs des successeurs ne sont corrects:** Dans ce cas, la recherche échoue. Le logiciel de couche supérieure utilisant Chord remarquera que les données n'ont pas été trouvées et réessaiera après une courte pause.

Cela conduit à la conclusion que les jonctions de nœuds n'ont aucun impact sur la correction, lorsqu'un grand nombre de nœuds se joignent entre la cible et le prédécesseur de la cible, les recherches peuvent être légèrement plus lentes, mais toujours correctes.

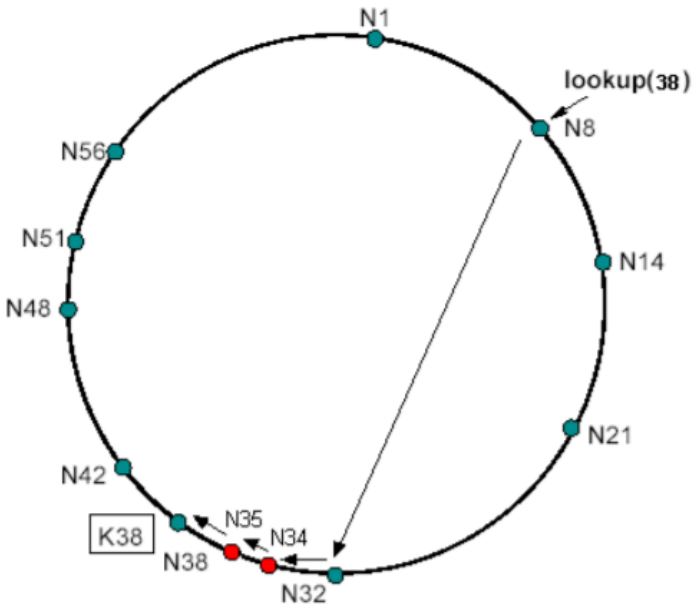


Figure 13: un grand nombre de nœuds se rejoignent entre la cible et le prédécesseur de la cible.

2. Impact des rejointes de nœuds sur les performances

Lorsque la stabilisation est terminée, il n'y a pas d'impact sur les performances au-delà de l'augmentation de N (nombre total de nœuds) dans le temps de consultation $O(log\ N)$ .Lorsque la stabilisation n'est pas terminée, la vitesse de consultation peut être affectée si des nœuds se joignent entre la cible et le successeur de la cible, comme mentionné dans le deuxième cas de la section 1 Mais ce n'est le cas que si le nombre de nœuds de jonction est très élevé. En général, on peut affirmer que les consultations prennent $O(log\ N)$ sauts tant que le temps nécessaire pour ajuster les tables de hachage est inférieur au temps nécessaire pour doubler la taille du réseau.

Défaillance des nœuds

L'exactitude du protocole Chord repose sur le fait que chaque nœud connaît son successeur. Lorsque des nœuds tombent en panne, il est possible qu'un nœud ne connaisse pas son nouveau successeur et qu'il n'ait aucune chance de l'apprendre. (La figure 14) montre un exemple.

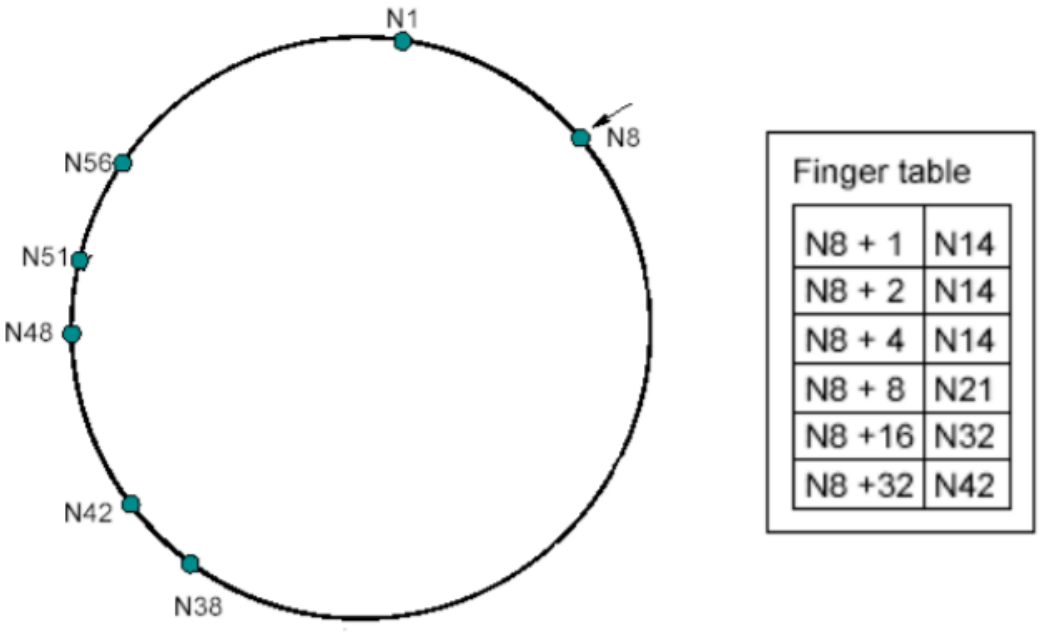


Figure 14: Un exemple de défaillance de nœuds

Lorsque les nœuds N14, N21 et N32 tombent simultanément en panne, N8 n'a aucune chance d'apprendre l'existence de son nouveau successeur N38, puisqu'il n'apparaît pas dans la table de hachage de N8.

Applications qui utilisant Chord

Les applications suivantes sont des exemples d'utilisation de Chord:

- **Mise en miroir coopérative**: Dans lequel plusieurs fournisseurs de contenu coopèrent pour stocker et servir les données des autres. En répartissant la charge totale de manière égale sur les hôtes de tous les participants, on réduit le coût total du système, puisque chaque participant ne doit fournir de la capacité que pour la charge moyenne, et non pour la charge de pointe de ce participant.
- **Stockage en temps partagé:** Pour les nœuds dont la connectivité est intermittente. Si quelqu'un souhaite que ses données soient toujours disponibles, mais que son serveur n'est qu'occasionnellement disponible, il peut proposer de stocker les données des autres lorsqu'il est connecté, en échange de quoi ses données seront stockées ailleurs lorsqu'il sera déconnecté. Le nom des données peut servir de clé pour identifier le nœud Chord (vivant) responsable du stockage de l'élément de données à un moment donné.
- **DNS basé sur les accords:** Le DNS fournit un service de consultation, avec les noms d'hôtes comme clés et les adresses IP (et autres informations sur les hôtes) comme valeurs. Chord pourrait fournir le même service en hachant chaque nom d'hôte à une clé [14]. Le DNS basé sur Chord ne nécessiterait aucun serveur spécial, alors que le DNS ordinaire repose sur un ensemble de serveurs racine spéciaux. Le DNS nécessite une gestion manuelle des informations de routage (enregistrements NS) qui permettent aux clients de naviguer dans la hiérarchie des serveurs de noms ; Chord maintient automatiquement l'exactitude des informations de routage analogues. Le DNS ne fonctionne bien que lorsque les noms d'hôtes sont structurés de manière à refléter les limites administratives ; Chord n'impose aucune structure de nommage. Le DNS est spécialisé dans la recherche d'hôtes ou de services nommés, tandis que Chord peut également être utilisé pour trouver des objets de données qui ne sont pas liés à des machines particulières.

Conclusion

Dans ce chapitre, nous avons présenté différents réseaux peer-to-peer, à savoir les réseaux P2P centralisés, hybrides et décentralisés, non structurés et structurés. Nous avons donné des illustrations qui montrent leurs topologies respectives, les forces et les inconvénients de chaque type. Nous avons donné les objectifs et les principales caractéristiques.

Ensuite nous avons parlé du Chord qu'est un protocole simple mais puissant qui résout le problème de la localisation efficace des données. Sa seule opération consiste à mettre en correspondance une clé avec le nœud responsable. Chaque nœud maintien des informations de routage sur $O(log\ N)$ autres nœuds, et les recherches sont réalisables via $O(log\ N)$ messages. Par conséquent, Chord s'adapte bien à un certain nombre de nœuds, ce qui en fait une application intéressante pour les grands systèmes. Chord continue à fonctionner correctement même si le système subit des changements majeurs et si les informations de routage ne sont que partiellement correctes.

CHAPITRE 3: SIMULATION ET REALISATION DU SIMULATEUR

Introduction

Dans la deuxième partie du chapitre 2, nous avons présenté le protocole Chord et les détails des différents algorithmes et des principales fonctions du protocole (rejointes des nœuds, défaillance des nœuds etc…).

Dans ce chapitre, nous allons présenter la définition de la simulation des réseaux peer-to-peer et les type des simulations d'une façon générale ensuite nous allons présenter notre solution en qui permet d'aider les développeurs à résoudre les problèmes liés à la création d'applications et de protocoles peer-to-peer.

La simulation des réseaux peer-to-peer

La simulation de réseau est un type de simulation de bas niveau. Les outils de simulation de réseau modélisent un réseau de communication en calculant le comportement de composants de réseau en interaction, tels que des hôtes et des routeurs, ou même des entités plus abstraites telles que des liaisons de données. Les outils de simulation de réseau permettent aux ingénieurs d'observer le comportement des composants du réseau dans des conditions spécifiques sans avoir à supporter les coûts de déploiement d'un réseau réel à grande échelle.

Les problèmes de conception de haut niveau pour l'infrastructure de communication numérique sont très difficiles. La grande échelle et l'hétérogénéité des applications, du trafic et des médias, combinées aux restrictions de qualité de service et au manque de fiabilité de la connectivité, en font un problème non trivial. Le développement d'applications et de protocoles au niveau du réseau implique un certain nombre de nœuds hétérogènes qui sont à la fois coûteux et difficiles à assembler. La simulation est donc la meilleure solution pour tester les applications et les protocoles de réseau de bas niveau.

La simulation peer-to-peer est une abstraction de la simulation générale de réseau. La simulation de protocoles peer-to-peer implique le transfert de messages entre pairs et la collecte de statistiques pertinentes pour la simulation.

La simulation peer-to-peer, comme la simulation de réseau générale, est composée de deux parties de code différentes. Le code du simulateur est responsable de l'exécution de la simulation, il crée les pairs, maintient la boucle de simulation principale et délivre des messages si nécessaire. Le code du protocole simulé est responsable de la logique particulière du protocole, c'est le code à exécuter lorsqu'un nœud doit être simulé. Ce code sera exécuté soit pour simuler l'arrivée de messages, soit à intervalles réguliers pendant la boucle principale.

La simulation de très grands réseaux est particulièrement importante pour la simulation de protocoles et de systèmes peer-to-peer. L'environnement de déploiement habituel d'une application peer-to-peer est un réseau étendu. La capacité d'un simulateur peer-to-peer à s'adapter à la taille d'un réseau étendu réel est un facteur très important dans le choix d'un simulateur peer-to-peer. Un simulateur peer-to-peer. Un autre facteur important est la qualité de la documentation du simulateur. Le simulateur doit être configuré à l'aide d'un langage de configuration déclaratif ou procédural, nous devons prendre en considération la facilité et la puissance du langage.

1. Type de simulations

La simulation est un outil important pour tester les protocoles, les applications et les systèmes en général. La simulation peut être utilisée pour fournir des données empiriques sur un système, simplifier la conception et améliorer la productivité, la fiabilité en évitant les coûts de déploiement. Les bancs d'essai de simulation offrent différents niveaux de sémantique/abstraction dans leur configuration et leur exécution selon le niveau d'abstraction souhaitable pour chaque type de simulation.

1.1. Simulation à événements discrets

Les simulations traditionnelles à événements discrets sont exécutées de manière séquentielle. Une variable $clock$ maintient l'état actuel de la simulation et est mise à jour au fur et à mesure de sa progression. Une structure de données $eventlist$ maintient un ensemble de messages dont la livraison est prévue dans le futur. Le message dont le délai de livraison est le plus proche est retiré de la liste des événements. La liste des événements, le processus correspondant est simulé et le $clock$ est mis à jour avec le délai de livraison. Si le processus simulé génère d'autres messages, ceux-ci sont

ajoutés à la liste des événements. Cette méthode est appelée simulation événementielle car l'horloge se déplace toujours vers l'heure de livraison suivante et jamais entre les deux.

1.2. Simulation en temps réel

La simulation en temps réel est née des environnements de formation virtuels. Particulièrement utile pour les militaires, elle s'attend à ce que le temps réel intègre des composants simulés avec des entités vivantes, telles que des humains. Elle souffre de problèmes d'extensibilité, car l'ensemble de la simulation et des activités simulées doit être exécuté en temps réel (probablement de manière concurrente).

1.3. Simulation basée cycle

La simulation basée cycle ressemble à la simulation en temps réel. Dans la simulation basée sur un cycle, chaque composant simulé (le pair) est exécuté une fois par cycle, qu'il ait ou non du travail à faire. Cette méthode offre une plus grande abstraction que le moteur basé sur les événements, car les informations relatives aux pairs simulés sont disponibles à tout moment de la simulation. Le niveau d'encapsulation lors de la simulation d'un pair individuel est laissé à l'appréciation du réalisateur du protocole.

Limites et solutions possibles

La simulation peer-to-peer actuelle souffre d'une limitation du nombre de pairs due à l'utilisation de la mémoire. Lors de la simulation sur un seul ordinateur, cette limitation ne peut pas être surmontée. D'autres approches, telles qu'un environnement de déploiement virtualisé, se sont avérées inefficaces et incapables de dépasser la limite de mémoire en utilisant des quantités raisonnables de ressources. Par conséquent, le besoin d'une solution sur mesure tenant compte des caractéristiques de mise en œuvre de la simulation d'égal à égal supprime la limite de mémoire et exécute toujours la simulation avec des performances acceptables.

1. Les simulateurs peer-to-peer existants

1.1 Peersim

Peersim [15] est un simulateur peer-to-peer écrit en Java. Il est publié sous licence GPL, ce qui le rend très attractif pour la recherche, attractif pour la recherche.

Peersim offre à la fois des moteurs basés sur les cycles et sur les événements. C'est le seul simulateur peer-to-peer dont il est question ici qui discuté ici qui offre un support pour le mode basé sur le cycle. Les auteurs de Peersim affirment que la simulation peut atteindre 10^6 nœuds dans ce mode.

Le mode basé sur le cycle est bien documenté avec des exemples, des tutoriels et une documentation au niveau de la classe. Le mode événementiel mode événementiel, par contre, n'est documenté qu'au niveau de la classe. Peersim utilise un langage simple et personnalisé pour la configuration de la simulation. Tout le contrôle et la collecte de statistiques doivent être effectués en étendant des classes du simulateur qui seront ensuite exécutées dans la simulation.

Peersim offre une certaine simulation de réseau sous-jacent dans le mode événementiel, il respectera le délai du message comme demandé par l'expéditeur, retard des messages comme demandé par l'expéditeur.

1.2 P2PSim

P2PSim [16] est un simulateur peer-to-peer qui se concentre sur la simulation du réseau sous-jacent. Il est écrit en C++ et, comme dans Peersim, les développeurs peuvent étendre les classes du simulateur pour mettre en œuvre des protocoles peer-to-peer.

La pile de simulation réseau fait de l'évolutivité un problème dans P2PSim. Les développeurs de P2PSim ont pu tester le simulateur avec jusqu'à 3000 nœuds.

La documentation C++ est pauvre mais existante. Des scripts d'événements peuvent être utilisés pour contrôler la simulation. A mécanisme minimal de collecte de statistiques est intégré au simulateur.

1.3 PlanetSim

PlanetSim [17] est également un simulateur à événements discrets écrit en Java. Le simulateur peut évoluer jusqu'à 10^5 nœuds. L'API et la conception ont été largement documentées. Le support pour la simulation du réseau sous-jacent est limité, cependant il est possible d'utiliser les informations BRITE [18] à cette fin.

1.4 QueryCycle

Le simulateur QueryCycle [19] est spécialisé dans les simulations de partage de fichiers. Il dispose de modèles réalistes pour la distribution de contenu, l'activité de requête, le comportement

de téléchargement, etc. La distribution du contenu est basée sur un modèle, où chaque fichier appartient à une catégorie et cette catégorie est définie par la popularité du fichier.

Les simulations se déroulent dans des cycles de requête représentant la période de temps entre l'émission d'une requête et le téléchargement. Période de temps entre l'émission d'une requête et la réception d'une réponse. Les requêtes générées sont passées dans une file d'attente et traitées selon le principe du "FIFO (first in first out)".

Le tableau ci-dessous présente un résumé de la revue bibliographique des différents simulateurs P2P.

Simulateur	Avantages	Inconvénients
P2PSim	puissante interface graphique intégrée pour l'observation, la collecte et l'analyse des données statistiques.	statistiques limitées, passage à l'échelle: 3000 nœuds
PeerSim	passage à l'échelle: 10^6 nœuds en mode cyclique	réseau physique non modélisé, Peu de documentation sur le mode événement discret
PlanetSim	passage à l'échelle: 10^5 nœuds	pas de statistiques, réseau physique non modélisé
QueryCycle	spécialisé dans les systèmes d'échange de fichiers, modélisation du réseau	mode cyclique, passage à l'échelle: 20 nœuds

Tableau 2: Caractéristiques techniques des Simulateurs existants

Présentation de notre simulateur

Comme nous avons souligné dans l'introduction notre solution est devisée en deux parties:

- La partie simulation: nous allons présentes dans cette section l'implémentation du simulateur basée cycle qu'on a proposé.
- La partie logique: dans laquelle nous définissons les fonctions du protocole Chord.

La simulation suit l'algorithme suivant:

- On fixe le nombre de nœuds du réseau.
- On fixe le nombre de cycle pour déterminer le temps de la simulation.
- Dans chaque cycle :
    - Tous les nœuds sont sélectionnés un par un de façon aléatoire (l'ordre de sélection est modifier a fin de chaque cycle) pour que chacun puis faire une opération parmi : départ, arrivée, recherche d'une ressource, publication d'une recherche ou aucune opération.
    - A chaque cycle un nombre de nouveaux nœud rejointe le réseau.
- A la fin de chaque cycle concernent quelques aspects qu'on analyser à la fin de la simulation

Voici le code responsable de l'initialisation de l'anneau de Chord:

```
// init chord ring
ChordRing chordRing = new ChordRing ();
for (int i = 0; i < init_node_num; i++) {

    URL url = new URL("http", host, port + i, "");
    Node newNode = new Node(url.toString());
    chordRing.addNode(newNode);
    }
chordRing.initRing();
System.out.println(chordRing.getCurNodeNum() + " nodes are created.");
chordRing.printAllNode();
```

Figure 15: code qui faire l'initialisation de l'anneau

Dans une boucle de 30 cycles chaque nœud dans l'anneau de Chord à une fonction aléatoire à exécuter: $leaveNode(), publishFile(), lookupForFile(), noOperation()$, ou lance la fonction $joinNode()$ aléatoirement pour intégrer le réseau.

- $leaveNode()$: cette fonction permet de nœuds de quitter le réseau.
- $publishFile()$: cette fonction permet à un nœud de publier une ressource, nous avons apte puis que la ressource publies fait partie d'une liste de 300 ressources connues par tous les nœuds ainsi lorsque un nœud veut publier, il choisit une ressource de cette liste.
- $lookupForFile()$: cette fonction permet au nœud de rechercher une ressource, la ressource recherchée peut être parmi la liste des ressources publiées ou autre (pour les ressources qui n'existent pas vraiment), on a deux cas à ce niveau
    - Échec: la ressource recherchée n'existe pas dans le réseau. ou bien le nœud qui publie cette ressource a quitté le réseau.
    - Succès: La ressource recherchée existe dans le réseau et le nœud qui a publié cette ressource est encore dans le réseau.
- $noOperation()$: Le nœud n'a pas d'opération à faire dans ce cycle.

L'algorithme suivant explique le déroulement de notre simulateur.

```
// start simulating
Random random = new Random ();
int cycleNumber = 1;
while (cycleNumber ⩽ 30){
    int x = random.nextInt(5);
    for (int k =0;k<x;k++){
        joinNode(chordRing);
    }
    for(int j = 0; j< ChordRing.nodesList.size(); j++){
        x = random.nextInt(4);
        if (x==0){
            publishFile(chordRing.getNode(j));
        }
        if (x==1){
            noOperation(chordRing.getNode(j));
        }
        if (x==2){
            leaveNode(chordRing.getNode(j));
        }
        if (x==3){
            lookupForFile(chordRing.getNode(j));
        }
    }
    cycleNumber++;
    ChordRing.nodesList.changeOrder();
}

```

Figure 16: La boucle principale du simulateur

A la fin de chaque cycle de la boucle on change l'ordre des nouds dans l'anneau pour un nouveau cycle et enregistre les mesures suivantes:

- La recherche de ressource est réussie.
- La recherche de ficher est échouée puisque cette ressource n'est pas publier.
- La rejointe d'un nouveau nœud est réussie.
- La rejointe d'un nouveau nœud est échouée.
- Le départ d'un nœud est réussi.
- Le départ d'un nœud est échoué.
- La publication d'une ressource est réussie.
- La publication d'une ressource est échouée.

Conclusion

Nous avons parlé dans ce chapitre des généralités sur la simulation des réseaux pair-à-pair et leur trois types: simulation basse cycle, simulation à évènement discrets et simulation en temps réel, ensuit la définition et la comparaison entre les meulier simulateurs pair-a-pair existe selon leur avantages.

Finalement la présentation de notre simulateur et le déroulement de cette simulation avec les différentes méthodes possibles pour chaque nœud dans l'anneau du Chord.

CHAPITRE 4: IMPLEMENTATION ET ANALYSE DES MESURES

Introduction

Après la présentation de la simulation et l'explication du déroulement dans le chapitre précèdent, nous allons présenter dans ce chapitre l'implémentation de Chord protocole et l'adaptation de ce protocole sur notre simulateur puis on va discuter est analyser les mesures obtenu à la fin de la simulation.

Implémentation du protocole Chord

1.1. Node

La classe Node contient les attributs suivants:

- id: identifiant du nœud.
- $hashKey$: valeur hachée en 32 bits génère par la fonction $cumputeHash()$ dans la classe Hash.
- $predecessor$: le nœud prédécesseur du nœud actuel.
- $successor$: le nœud successeur du nœud actuel.
- $fingerTable$: la table de hachage pour chaque nœud
- $files$: liste des fiches partagées dans le réseau pour le nœud actuel

```
1 public Node (String nid) {
2
3     this.nid = nid;
4     this.hashKey = new HashKey (nid);
5     this.predecessor = this;
6     this.successor = this;
7     this.fingerTable = new FingerTable(this);
8     this.valid = true;
9     this.files = new ArrayList<> ();
10 }
```

Figure 17: Création d'un nœud

La méthode $stabilize()$ dans cette classe qui vérifie périodiquement et immédiatement le successeur de n puis elle informe ce dernier.

```java
public void stabilize () {

    Node newAdd = this.successor.predecessor;
    if (newAdd != null && newAdd.checkValid()) {
        HashKey hashKey = newAdd.hashKey;
        // if key of oldPre in interval (this, successor)
        if (hashKey.inCurInterval(this.hashKey, this.successor.hashKey)) {
            this.successor = newAdd;
            this.secondSuccessor = newAdd.successor;
            this.predecessor.secondSuccessor = this.successor;
        }
    } else {
        if (!newAdd.checkValid()) {
            this.secondSuccessor = this.successor.successor;
            this.predecessor.secondSuccessor = this.successor;
        }
    }
}   this.successor.notify(this);
```

Figure 18: La fonction stabilize()

La fonction $lookup()$ dans la figure ci-dessous fait la recherche dans le réseau sur le nœud qui a la ressource, qui accepte deux paramètres, la valeur hachée de la ressource à chercher et une valeur booléenne pour déterminer à chaque fois le nœud actuel et retourner le nœud qui a cette ressource.

```java
public Node lookup (HashKey hashKey,boolean verbose) {

    if (verbose) {
        System.out.println("Tracing... " + this.nid);
    }
    if (hashKey.inCurInterval(this.getHashKey (), this.getSuccessor().getHashKey ())) {
        return this.successor;
    }
    Node temp = this.getFinger(hashKey);
    if (temp.hashKey.toDecimal() == hashKey.toDecimal()) {
        return temp;
    } else {
        return temp.lookup(hashKey, simple, verbose);
    }
}
```

Figure 19: La fonction lookup()

1.2. FingerTable

La classe FingerTable contient l'attribut suivant:

- $fingers$: liste des fingers une instance de la classe Finger qui contient deux attributs une valeur hachée $strat$ et $successeur$.

Le constructeur de cette classe initialise la table de hachage pour le nœud actuel:

```
1 public FingerTable(Node node) {
2
3     fingers = new ArrayList<Finger>();
4     // initiate the finger table
5     for (int i = 0; i < Hash.getHashLength(); i++) {
6
7         HashKey start = node.getHashKey().createStart(i);
8         Node successor = node.lookup(start, true, false);
9     }   fingers.add(new Finger(start, successor));
10 }
```

Figure 20: Constructeur de la classe FingerTable

1.3. Hash

La classe Hash contient les attributs suivants:

- $BYTE\_LENGHT$: une constante pour stocker la taille d'octet en bits.
- $hash\_lenght$: la taille de la valeur après le hachage.
- $bnum$: le nombre de bits.

La méthode $setLengthByte()$ définir la longueur de la clé de hachage en fonction du nombre maximal de nœuds:

```
1 public static void setLengthInByte(int node_num) {
2
3     hash_length = Math.max(hash_length, (int) Math.ceil(Math.log(node_num)
4     / Math.log(2)));
5
6     bnum = (int) Math.ceil(hash_length / (double)BYTE_LENGTH);
7 }   System.out.println("HashKey size = " + bnum + "Bytes");
```

Figure 21: La méthode setLengthByte()

La fonction $computeHash$ pour hacher la valeur hachée utilisant SHA-1:

```java
public static byte[] computeHash(String input) {

    byte[] value = null;
    try {
        // calculate initial SHA-1 hash
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        md.reset();
        byte[] init_code = md.digest(input.getBytes());
        value = new byte[bnum];
        int shrink = init_code.length / value.length;
        int count = 1;
        // shrink the SHA-1 hash to bnum size
        for (int i = 0; i < init_code.length * BYTE_LENGTH; i++) {
            int cur = ((init_code[i / BYTE_LENGTH] & (1 << (i % BYTE_LENGTH))) >> i% BYTE_LENGTH);
            if (cur == 1) {
                count++;
            }
            if (((i + 1) % shrink) == 0) {
                int temp = (count % 2 == 0) ? 0 : 1;
                count = 1;
                value[i / shrink / BYTE_LENGTH] |= (temp << ((i / shrink) % BYTE_LENGTH));
            }
        }
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    return value;
}
```

Figure 22: La fonction computeHash()

1.4. ChordRing

La classe ChordRing contient l'attribut suivant:

- nodeList: la liste des nœuds existants dans l'anneau de Chord.

La méthode $join()$ est définie dans cette classe, elle accepte un nouveau nœud pour faire la rejointe de ce nœud à l'anneau.

```java
public void join(Node newNode) {

    newNode.setPredecessor(null);
    Node successor = nodeMap.get(nodeHeap.peek()).lookup(newNode.getHashKey(),

    this.SIMPLE, false);
    newNode.setSuccessor(successor);
    newNode.setSecondSuccessor(successor.getSuccessor());
    nodeHeap.add(newNode.getHashKey().toHex());
    nodeMap.put(newNode.getHashKey().toHex(), newNode);
}
```

Figure 23: La méthode join()

La méthode $leave()$ est définie dans cette classe, elle accepte un nœud pour quitter le système.

```
public void leave (Node node) {

    for (File file : node.getFiles ()) {
        file.hashKey=null;
        file.published = false;
    }
    node.fail();
    nodeMap.remove(node.getHashKey().toHex());
    nodeHeap.remove(node.getHashKey().toHex());
}   initList.remove (node);
```

Figure 24: La méthode leave()

La fonction $getNode()$ est définit dans cette classe pour trouver le nœud actuelle qui fait une des opérations possible:

```
Node getNode(int index ){

    return initList.get (index);
}
```

Figure 25: La fonction getNode()

Analyse des Mesures

Avec l'utilisation de classe Logger qui permet d'enregistrer les résultats dans un fichier externe output.txt (Annexe A):

```
class Logger {
    static void log (String message) throws IOException {
        PrintWriter out = new PrintWriter(new FileWriter ("output.txt", true), true);
        out.write(message+"\n");
        out.close();
    }
}
```

Figure 26: La classe Logger

1.1. Résultats pour chaque cycle

Exemple d'un cycle aléatoire qui affiche les résultats et les mesures lorsqu'on simule 1000 nœuds:

```
1 ---------------------- Cycle: 23 Simulation Results ----------------------------
2 Search File Failure : 16 from total of : 193 search operations
3 Search File Success : 177 from total of : 193 search operations
4 800 nodes in the ring
5
```

Figure 27: Résultat de la simulation pour un cycle

Les remarques qu'on peut extraire de ces résultats:

- Dans ce cycle le système fait 193 opérations de recherche d'une ressource sur le réseau.
    - Il trouve la ressource dans 177 cas avec succès.
    - Il n'a pas trouvé la ressource dans 16 cas car cette ressource n'existe pas dans le réseau ou bien le nœud qui publie la ressource a quitté le réseau.

1.2. Résultats pour tous les cycles

Les résultats et les mesures lorsqu'on simule 1000 nœuds en 30 cycles:

```
1 ---------------------- Simulation Results ----------------------------
2 Failure : 195 from total of 5916 search operations
3 Success : 5721 from total of 5916 search operations
4 Failure : 0 from total of 321 join operations
5 Success : 321 from total of 321 join operations
6 Failure : 0 from total of 274 leave operations
7 Success : 274 from total of 274 leave operations
8 Failure : 0 from total of 274 publish operations
9 Success : 5960 from total of 274 publish operations
```

Figure 28: Résultat de la simulation pour tous les cycles

Représentation graphiques

Pour la représentation graphique de nos résultats obtenus durant la simulation, nous avons utilisés la bibliothèque "matplotlib" de python.

Une simulation en 30 cycles et pour 1000 nœuds, nous avons obtenu les résultats suivants:

- Le système fait entre 150 et 300 recherches réussites d'une ressource par cycle.
- Le système fait entre 0 et 20 recherches échouées d'une ressource par cycle.

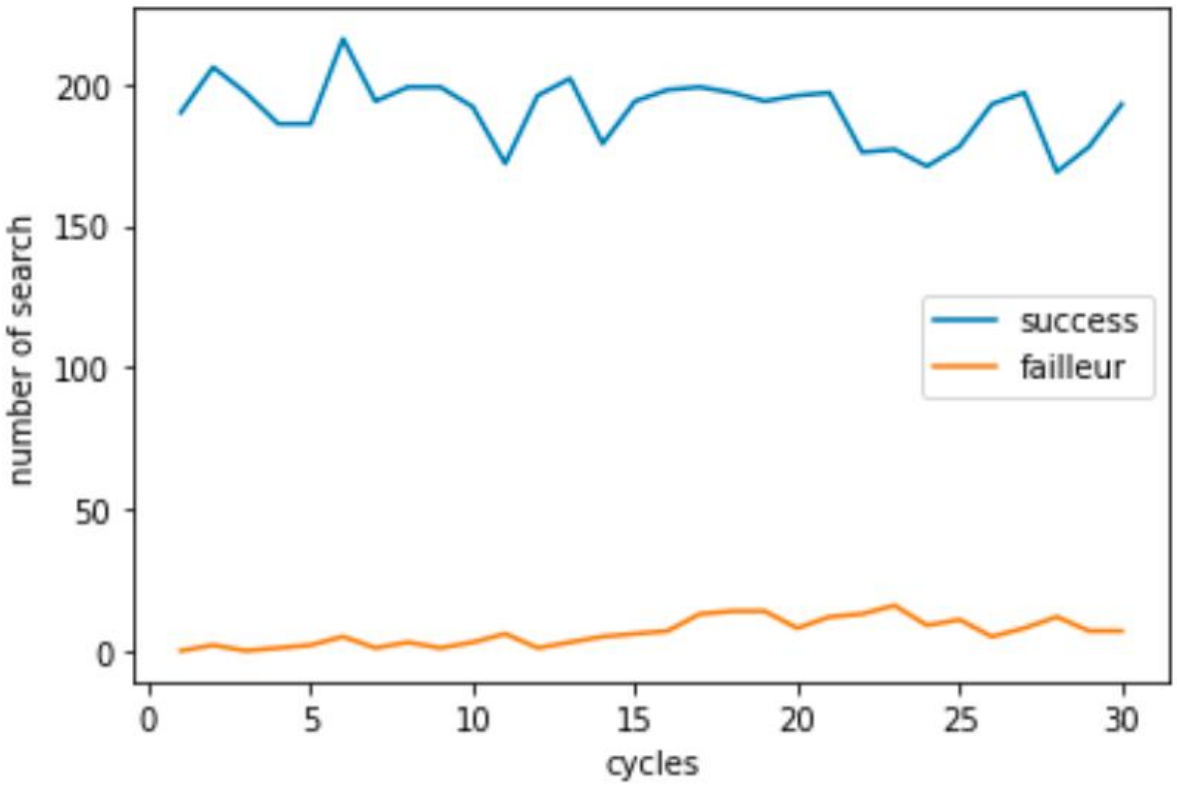


Figure 29: Représentation graphique des résultats de recherches de ressources pour chaque cycle

A la fin de l'exécution de 30 cycles on sorte avec les résultats suivants (Figure 30, Figure 31):

- Le système fait au total 5721 recherches réussites d'une ressource.
- Le système fait au total 195 recherches échouées d'une ressource.

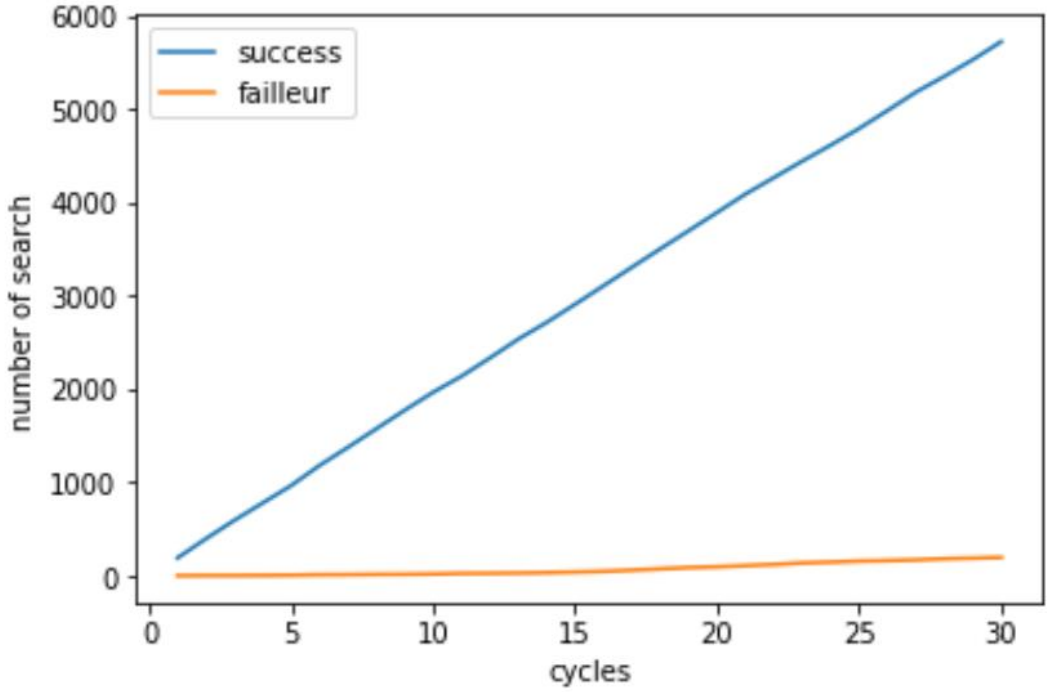


Figure 30: Représentation graphique des résultats de recherches de ressources pour les 30 cycles (courbe)

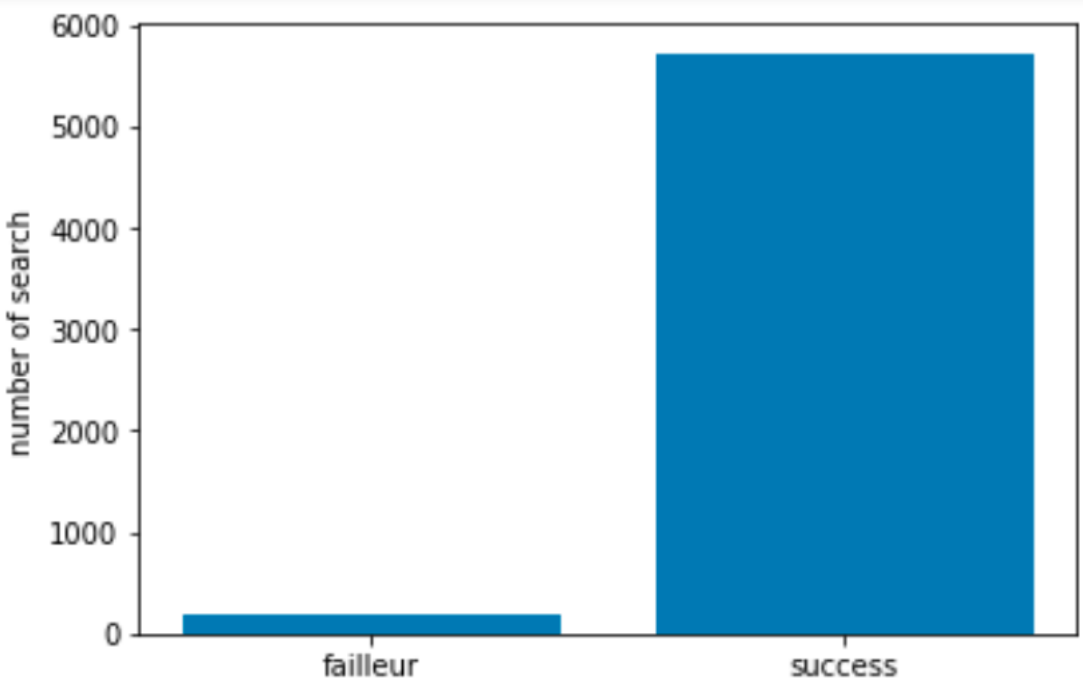


Figure 31:Représentation graphique des résultats de recherches de ressources pour les 30 cycles (histogramme)

Conclusion

Dans ce chapitre nous avons mis le point sur l'implémentation du protocole Chord avec une présentation de code source pour expliquer les algorithmes qu'on a implémenté et après l'adaptation de ce protocole puis nous avons réalisé une présentation graphique des résultats obtenus à partir du simulateur avec une analyse et Interprétation des résultats.

CONCLUSION GENERALE

Le travail des simulateurs spécifiques se limite à l'analyse d'un environnement totalement simule. Leur inflexibilité les empêche de simuler des appareils dans un environnement partiellement simulé, c'est-à-dire ou des nœuds du réseau simulé sont des ordinateurs existants physiquement équipés d'une application capable d'interpréter le protocole testé. Des applications déjà fonctionnelles peuvent être mises dans un contexte simule sans avoir à être elles-mêmes placées dans un environnement simulé.

A l'issu de ce travail qui porte sur la simulation et l'analyse d'un réseau peer-to-peer structuré avec DHT basée sur Chord, vu que l'importance de mis en place un environnement de test pour évaluer les caractéristiques du protocole, nous avons réalisé un simulateur basée cycle qui simule ce type des réseaux avec plus de 1000 nœuds pour trente 30 cycles et extraire des mesures pour chaque cycle et aussi à la fin de la simulation. Et nous avons pu répondre à la problématique posé dans le premier chapitre ainsi que réalisé nos objectives qui sont:

- Proposer et d'implémenter une simulation dirigée par le cycle d'une architecture de réseau peer-to-peer basée sur Chord.
- Faire des analyses des résultats mesurées.

Ce projet simule seulement la recherche en utilisant Chord, aucune connexion réelle ou transfert de ressource n'a été établi.

Parmi nos perspectives est d'ajouter une Interface graphique permettant de visualiser le comportement des nœuds au sein de réseau en temps réel.

REFERENCES

[1] R. M. D. K. M. K. a. H. B. I. Stoica, «Chord: A scalable peer-to-peer lookup service for internet applications,» chez *Proceeding of SIGCOMM*, 2001.

[2] S. R, «A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications,» chez *in Conference on Peer-to- Peer Computing*, Sweden, 2001.

[3] S. A.-T. a. D. Spinellis, «A survey of peer-to-peer content distribution technologies,» chez *ACM Computing Surveys*, 2004.

[4] R. A. KING, «Localisation de sources de données et optimisation de requêtes réparties en environnement pair-à-pair,» chez *2010*, Toulouse.

[5] Y.-K. R. Kwok, «Peer-to-Peer Computing,» chez *CRC Press*, 2011.

[6] B. T. a. S. V. N. Budan, «New Technologies Networks Peer To-Peer Networks, Computer Network Engineer Thesis,» 2003.

[7] P. F. E. B. G. U.-K. K. R. L. Garces-Erice, «Data Indexing in Peer-to-Peer DHT Networks,» 2003.

[8] D. I. J. O. U. X. Alexandre et E. M. M. Samuel, « Peer-to-peer,» chez *Université Claude Bernard*, LYON, 2006.

[9] N. I. o. S. a, «Technology, Secure Hash Standard,» chez *FIPS*, 1995.

[10] H. Y. E. K. L. John Buford, «P2P Networking and Applications,» Morgan Kaufmann, 2009, pp. 30-32.

[11] M. R. J. Buford, «Instant Messaging and Presence Service (IMPS),» chez *A. Salkintzis*, 2005.

[12] R. D. Williams, «Parallel Computing Works,» chez *Kaufmann Publishers*.

[13] D. L. E. L. F. L. M. L. D. P. R. KARGER, «Consistent hashing and random trees: Distributed,» El Paso, 1997, p. 654–663.

[14] R. M. A. A. M. R. COX, «Serving DNS using Chord," in In First International Workshop on Peer-to-Peer Systems,» Cambridge, 2002.

[15] A. M. a. M. Jelasity, «Peersim: A scalable p2p simulator. In Peer-to-Peer Computing,» *IEEE,* p. 99–100, 2009.

[16] F. K. J. L. R. M. a. J. S. T. Gil, «p2psim, a simulator for peer-to-peer protocols,» 2003.

[17] C. P. R. M. J. P. H. T. a. R. R. P. Garcia, «Planetsim: A new overlay,» chez *Software Engineering and Middleware*, 2005.

[18] A. L. I. M. a. J. B. A. Medina, «BRITE: An approach to universal topology generation In Modeling, Analysis and Simulation of Computer and Telecommunication Systems,» *IEEE,* p. 346–353, 2001.

[19] C. T. a. K. S. Schlosser M., «Simulating a P2P File-Sharing Network", 1st Workshop on Semantics in Grid and P2P Networks,» 2002.

WEBOGRAPHIES

|SETI| Seti@home, https://setiathome.berkeley.edu/

|ENTROPIA| entropia, https://www.entropia.com/

|IDRIVE| I-drive Cloud Backup, https://www.idrive.com/

|Groove| Groove, https://www.groove.net/

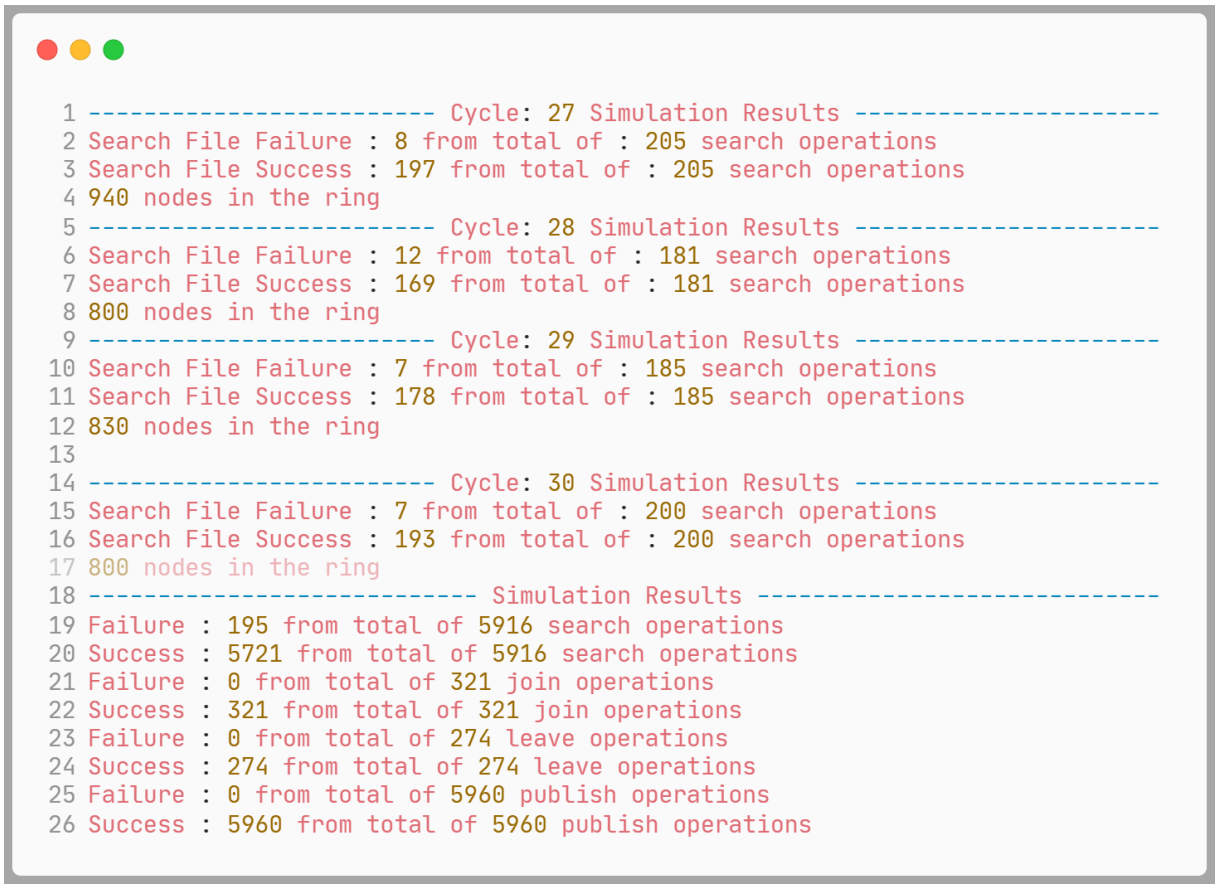
|NAPSTER| The ultimate digital music service "Napster", https://www.napster.com/

|GNUTELLA| Gnutella community, https://www.gnutella.com/

ANNEXE A

```
---------------------- Cycle: 1 Simulation Results ------------------------
Search File Failure : 0 from total of : 190 search operations
Search File Success : 190 from total of : 190 search operations
806 nodes in the ring
---------------------- Cycle: 2 Simulation Results ------------------------
Search File Failure : 2 from total of : 208 search operations
Search File Success : 206 from total of : 208 search operations
800 nodes in the ring
---------------------- Cycle: 3 Simulation Results ------------------------
Search File Failure : 0 from total of : 197 search operations
Search File Success : 197 from total of : 197 search operations
800 nodes in the ring
---------------------- Cycle: 4 Simulation Results ------------------------
Search File Failure : 1 from total of : 187 search operations
Search File Success : 186 from total of : 187 search operations
800 nodes in the ring
---------------------- Cycle: 5 Simulation Results ------------------------
Search File Failure : 2 from total of : 188 search operations
Search File Success : 186 from total of : 188 search operations
800 nodes in the ring
---------------------- Cycle: 6 Simulation Results ------------------------
Search File Failure : 5 from total of : 221 search operations
Search File Success : 216 from total of : 221 search operations
800 nodes in the ring
---------------------- Cycle: 7 Simulation Results ------------------------
Search File Failure : 1 from total of : 195 search operations
Search File Success : 194 from total of : 195 search operations
800 nodes in the ring
---------------------- Cycle: 8 Simulation Results ------------------------
Search File Failure : 3 from total of : 202 search operations
Search File Success : 199 from total of : 202 search operations
800 nodes in the ring
---------------------- Cycle: 9 Simulation Results ------------------------
Search File Failure : 1 from total of : 200 search operations
Search File Success : 199 from total of : 200 search operations
800 nodes in the ring
---------------------- Cycle: 10 Simulation Results ------------------------
Search File Failure : 3 from total of : 195 search operations
Search File Success : 192 from total of : 195 search operations
800 nodes in the ring
---------------------- Cycle: 11 Simulation Results ------------------------
Search File Failure : 6 from total of : 178 search operations
Search File Success : 172 from total of : 178 search operations
800 nodes in the ring
---------------------- Cycle: 12 Simulation Results ------------------------
Search File Failure : 1 from total of : 197 search operations
Search File Success : 196 from total of : 197 search operations
800 nodes in the ring
---------------------- Cycle: 13 Simulation Results ------------------------
Search File Failure : 3 from total of : 205 search operations
Search File Success : 202 from total of : 205 search operations
800 nodes in the ring
```

```
---------------------- Cycle: 14 Simulation Results ----------------------
Search File Failure : 5 from total of : 184 search operations
Search File Success : 179 from total of : 184 search operations
800 nodes in the ring
---------------------- Cycle: 15 Simulation Results ----------------------
Search File Failure : 6 from total of : 200 search operations
Search File Success : 194 from total of : 200 search operations
800 nodes in the ring
---------------------- Cycle: 16 Simulation Results ----------------------
Search File Failure : 7 from total of : 205 search operations
Search File Success : 198 from total of : 205 search operations
800 nodes in the ring
---------------------- Cycle: 17 Simulation Results ----------------------
Search File Failure : 13 from total of : 212 search operations
Search File Success : 199 from total of : 212 search operations
800 nodes in the ring
---------------------- Cycle: 18 Simulation Results ----------------------
Search File Failure : 14 from total of : 211 search operations
Search File Success : 197 from total of : 211 search operations
800 nodes in the ring
---------------------- Cycle: 19 Simulation Results ----------------------
Search File Failure : 14 from total of : 208 search operations
Search File Success : 194 from total of : 208 search operations
800 nodes in the ring
---------------------- Cycle: 20 Simulation Results ----------------------
Search File Failure : 8 from total of : 204 search operations
Search File Success : 196 from total of : 204 search operations
800 nodes in the ring
---------------------- Cycle: 21 Simulation Results ----------------------
Search File Failure : 12 from total of : 209 search operations
Search File Success : 197 from total of : 209 search operations
800 nodes in the ring
---------------------- Cycle: 22 Simulation Results ----------------------
Search File Failure : 13 from total of : 189 search operations
Search File Success : 176 from total of : 189 search operations
800 nodes in the ring
---------------------- Cycle: 23 Simulation Results ----------------------
Search File Failure : 16 from total of : 193 search operations
Search File Success : 177 from total of : 193 search operations
800 nodes in the ring
---------------------- Cycle: 24 Simulation Results ----------------------
Search File Failure : 9 from total of : 180 search operations
Search File Success : 171 from total of : 180 search operations
800 nodes in the ring
---------------------- Cycle: 25 Simulation Results ----------------------
Search File Failure : 11 from total of : 189 search operations
Search File Success : 178 from total of : 189 search operations
800 nodes in the ring
---------------------- Cycle: 26 Simulation Results ----------------------
Search File Failure : 5 from total of : 198 search operations
Search File Success : 193 from total of : 198 search operations
800 nodes in the ring
```



```
------------------------- Cycle: 27 Simulation Results ----------------------
Search File Failure : 8 from total of : 205 search operations
Search File Success : 197 from total of : 205 search operations
940 nodes in the ring
------------------------- Cycle: 28 Simulation Results ----------------------
Search File Failure : 12 from total of : 181 search operations
Search File Success : 169 from total of : 181 search operations
800 nodes in the ring
------------------------- Cycle: 29 Simulation Results ----------------------
Search File Failure : 7 from total of : 185 search operations
Search File Success : 178 from total of : 185 search operations
830 nodes in the ring

------------------------- Cycle: 30 Simulation Results ----------------------
Search File Failure : 7 from total of : 200 search operations
Search File Success : 193 from total of : 200 search operations
800 nodes in the ring
---------------------------- Simulation Results -----------------------------
Failure : 195 from total of 5916 search operations
Success : 5721 from total of 5916 search operations
Failure : 0 from total of 321 join operations
Success : 321 from total of 321 join operations
Failure : 0 from total of 274 leave operations
Success : 274 from total of 274 leave operations
Failure : 0 from total of 5960 publish operations
Success : 5960 from total of 5960 publish operations
```