République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université AKLI MOHAND OULHADJ de Bouira
Faculté des Sciences et des Sciences Appliquées
Département mathématiques

Mémoire

En vue de l'obtention du Diplôme de MASTER

Filière : *mathématiques appliquées*

Spécialité : *Recherche opérationnelle*

Thème

Analyse des performances réseau pair à pair structurée

Réalisé par :

SLIMANI Mohammed Riadh

Soutenu le 03 juillet devant le jury composé de :

M^{me}. OUIDJA Daya	MAA	Présidente	UAMO Bouira
M^{r}. HAMDOUNI Omar	MAA	Promoteur	UAMO Bouira
M^{r}. KHALDI Laala	MCA	Examinateur	UAMO Bouira
M^{r}. BIROUCHE madjid	MAA	Examinateur	UAMO Bouira

2023/2024

Remerciements

Je remercie Allah, le créateur tout-puissant, d'avoir guidé mes pas vers la connaissance et éclairé mon chemin. Je lui suis également reconnaissant de m'avoir donné le courage et la persévérance nécessaires pour mener mon travail à bien.

Je tiens à remercier mon directeur de recherche, M^r. *HAMDOUNI Omar*, qui s'est montré disponible pour me guider avec des conseils et des commentaires rigoureux.

Je tiens aussi à remercier M^{me} *OUIDJA Daya* d'accepter de présider le jury de soutenance.

Je remercie également, M^r *BIROUCHE Madjid* et M^r *KHALDI Laala* qui m'ont fait l'honneur d'accepter d'évaluer mon travail.

Enfin,mes remerciements à tous mes enseignants qui m'ont enseigné et accompagné durant tout mon cursus.

Dédicaces

Avec l'expression de ma reconnaissance, Je dédie ce modeste travail ;

A ma famille, elle qui m'a doté d'une éducation digne, son amour a fait de moi ce que je suis aujourd'hui .
Particulièrement à mon très cher père, pour son soutien, son affection et la confiance qu'il m'a accordé .
A ma très chère mère, quoi que je fasse ou que je dise, je ne saurai point te remercier comme il se doit .
Ton affection me couvre, ta bienveillance me guide et ta présence à mes côtés a toujours été ma source de force pour affronter les différents obstacles. Que Dieu la protège .
A mes adorables sœurs : Sarah, Maha, Dounia, qui n'ont pas cessée de m'encourager et soutenir tout au long de mes études .
A mes nièces et neveux : Elyne, Iyad, Kenzi .
A mon ami Ilyes et tous mes amis, et tous ce qui me sont chers et je n'ai pas cité leurs nom .
A la personne qui a toujours été à mes côtés et qui me complète.

Résumé

L'étude des performances des réseaux pair à pair (P2P) revêt une importance cruciale pour assurer leur efficacité, leur robustesse et leur scalabilité. Les simulations jouent un rôle essentiel en offrant une compréhension approfondie du comportement de ces réseaux dans des environnements évolutifs. En intégrant modèles analytiques et simulations numériques, il devient possible de concevoir et d'optimiser des réseaux P2P afin de maintenir des performances optimales face à des défis complexes et variés.

Mots clé : Réseaux pair à pair, Chaîne de Markov, Réseau de Petri, Analyse des performances.

Abstract :

Performance analysis of peer-to-peer (P2P) networks is crucial for ensuring their efficiency, robustness, and scalability.

Analyzing the performance of peer-to-peer (P2P) networks is crucial to ensure their efficiency, robustness, and scalability. Simulations play a pivotal role in gaining deep insights into network performance under evolving environments. By combining analytical models and numerical simulations, it becomes feasible to design and optimize P2P networks capable of maintaining optimal performance in the face of diverse and complex challenges.

Keywords : Peer-to-peer networks, Markov chain, Petri net, performance analysis

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Les réseaux, en informatique, sont des ensembles d'ordinateurs et d'autres appareils éléctroniques interconnectés, permettant l'échange des données et des ressources. Ils peuvent être classifiés de diverses manières, notamment par leur architecture : centralisée, décentralisée, distribuée.

Parmi les réseaux distribués, les réseaux de type pair à pair se sont transformés en un modèle de communication indispensable dans le domaine des réseaux informatiques, grâce à leur capacité à distribuer des ressources et des tâches entre les nœuds sans la nécessité d'un serveur centralisé. Contrairement à l'architecture traditionnelle Client/Serveur, où les clients dépendent d'un serveur unique pour accéder aux ressources, les réseaux $P2P$ attribuent le rôle du client et le rôle du serveur pour chaque nœud. Ainsi, cette décentralisation améliore la résilience du réseau, réduit les coûts d'infrastructure et permet une meilleure utilisation des ressources disponibles.

Pour évaluer et optimiser les performances des réseaux $P2P$, il est important d'utiliser des outils de modélisation et d'analyse efficaces. En 1960, Carl Adam Petri a introduit les réseaux de Petri (RdP). Par définition, un réseau de Petri est un modèle mathématique utilisé pour décrire et analyser les processus concurrents, les interactions complexes et les systèmes distribués. Il se compose de places (représentant les états), de transitions (représentant les événements) et d'arcs (représentant les relations entre les états et les événements).

Dans le cadre d'analyse des réseaux P2P, les RdP offrent plusieurs avantages. Ils permettent de modéliser la dynamique des systèmes $P2P$, de simuler différents scénarios de charge et de congestion, et d'identifier les goulots d'étranglement et les points de défaillance potentiels. En outre, les RdP peuvent être étendus pour inclure des aspects temporels et stochastiques, ce qui est crucial pour représenter les comportements réels des réseaux $P2P$. L'utilisation des RdP dans la mesure des performances des réseaux $P2P$ ouvre des perspectives nouvelles pour l'optimisation et l'amélioration continue de ces systèmes complexes.

Dans ce contexte, l'objectif de ce mémoire est d'analyser les performances d'un réseau Pair à Pair en exploitant les RdP.

Ce mémoire s'articulera autour de quatre chapitres. Le premier chapitre sera consacré à une étude générale sur le réseau de Petri. Le deuxième chapitre abordera des généralités sur les réseaux pair à pair $P2P$. Finalement, le troisième chapitres, détaillera les méthodes d'évaluation des performances des systèmes $P2P$ et des réseaux des files d'attente. Enfin, le dernier chapitre portera sur l'application développée dans ce cadre.

Je conclue ce mémoire par une conclusion générale et une liste de références.

Chapitre 1

Réseau de Petri

1.1 Introduction

Les réseaux de Petri (RdP) ont été introduits dans les années 1960 par Carl Adam Petri comme un outil pour l'analyse et la modélisation des systèmes complexes. Initialement utilisés pour représenter les systèmes concurrents, les RdP sont depuis devenus un outil graphique puissant pour évaluer les performances de divers systèmes. Leur capacité à représenter des systèmes constitués de sous-systèmes parallèles, interagissant et partageant des ressources, les rend particulièrement utiles. Les RdP sont couramment utilisés dans divers domaines [5] . Une caractéristique essentielle des RdP est leur extensibilité, permettant des adaptations spécifiques aux besoins particuliers. Outre leurs capacités analytiques, les RdP offrent également des analyses qualitatives.

Les RdP permettent deux niveaux de modélisations : un niveau décrivant la structure des systèmes, les actions diverses et possibles, ainsi que chaque dépendance entre les différentes parties du système et les effets des conditions nécessaires pour le fonctionnement du système, c'est le niveau statique ; le niveau dynamique, en revenche, sert à décrire les comportements divers et possibles d'un système en le modélisant. A cet égard, ce chapitre portera sur les définitions, conceptes et propriétés principales des RdP.

1.2 Notions de Base d'un réseau de Petri

Les RdP sont des graphes orientés constitués d'un ensemble de places (P) et d'un ensemble de transitions (T). Une place représente une ressource, où une transition représente un évènement. Le marquage du graphe d'un réseau de Petri représente l'état du système décrit. [4]

1.2.1 Architecture d'un Réseau de Petri

Un réseau de Petri dit non marqué est décrit par un 4-uplet :

$$R = (P, T, Post, Pre); \tag{1.1}$$

où :
• $P = \{p_i, i \in [1, n]\}$, ensemble de places fini et non vide.
• $T = \{t_i, i \in [1, m]\}$, ensemble de transitions fini et non vide.
• $Post : PXT \longrightarrow \mathbb{N}$, l'application des incidences arrières (Post Places)

• $Pre : PXT \longrightarrow \mathbb{N}$, l'application des incidences avants (Previous Places)
On considère C la matrice d'incidence du RdP qui est définie comme suit :

$$C = Post - Pre;$$

Par définition ses fonctions C, $Post$ et Pre sont présentées à l'aide des matrices, dont le nombre de transitions correspond au nombre de colonnes, et le nombre de place correspnd au nombre de lignes.

Une place est représentée par un cercle et une transition par un rectangle dans un réseau de Petri. Ces derniers sont visualisés à l'aide d'un graphe orienté défini prédédemment. Les arcs relient les transitions et les places. Un arc sortant d'une place à une transition défini une précondition ($Pre(p,t) \not\equiv 0$), où un arc sortant d'une transition vers une place défini une postcondition ($Post(p,t) \not\equiv 0$). Généralement, les arcs sont pondérés avec une valeur par défaut de 1, bien qu'ils puissent avoir des valeurs supérieures, dans ce cas, la valeur doit être notée sur l'arc en question. Un réseau de Petri est dit ordinaire si et seulement si on trouve une pondération de 1 pour tout arc de son ensemble d'arcs.

1.2.2 Réseau de Petri marqué

Un réseau de Petri dit marqué est, par définition, le couple :

$$N = (R, M_0); \tag{1.2}$$

•R : un RdP.
•M_0 : marquage de base (intial), représenté par l'application :

$$M_0 : P \longrightarrow \mathbb{N}$$

$$p \longrightarrow M_0(p)$$

avec $M_0(p)$ représente le nombre de jetons présents dans la place p.

Illustration :

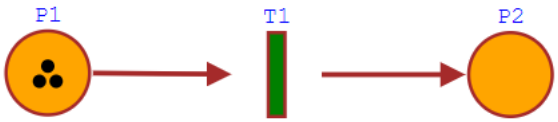

FIGURE 1.1 – Un réseau de petri marqué

Cette illustration montre un réseau de Petri avec son marquage. p_1 et p_2 des places qui contiennent un nombre entier non négatif de marques, également appelées jetons. $M_0(p_1) = 3$ est le nombre de jetons dans p_1, $M_0(p_2) = 0$ est le nombre de jetons dans p_2 . Le vecteur des marquages d'un réseau entier défini le marquage M de ce dernier.

$$M = (\, M(p_1), M(p_2)\,) = (3, 0)$$

Le marquage M est décrit par un vecteur dont la dimension correspond au nombre de places.

L'évolution du système modélisé, après la réalisation de certains événements, se traduit par le franchissement des transitions dans le marquage d'un RdP.

Proposition 1.2.1. *•Les places dans un RdP peuvent être soit occupées(marquées), soit vides(non marquées).*
•Une transition dans un RdP est considérée comme validée ssi chacune des places de son ensemble d'entrée est marquée.
•Les fonctions Pre et $Post$ déterminent le poids d'un arc.

1.3 Étude d'un Réseau de Petri

Pour appréhender un RdP, il est crucial de traduire le modèle mathématique en concepts concrets, en reliant les places, les transitions et les jetons à des éléments du monde réel.

1.3.1 Jeu de Transitions

Le jeu de transition permet de définir l'évolution d'un réseau de Petri. Une transition peut être tirée si et seulement si toutes ses places d'entrée contiennent un nombre de jetons supérieur ou égal au poids de l'arc qui les relie à la transition.
Lorsqu'une transition est tirée, deux choses se produisent :
•Des jetons sont retirés des places d'entrée de la transition.
•Des jetons sont ajoutés aux places de sortie de la transition.
Le nombre de jetons dans chaque place représente l'état du système à un instant donné.
Les règles de tir définissent le comportement dynamique du système. Elles permettent de simuler l'évolution du système en fonction des événements qui se produisent [3]

Définition 1.3.1 (franchissement). *On dit qu'une transition t est considérée comme franchissable, si chaque place p en entrée contient un nombre de jetons au moins égal à la valuation de l'arc qui la relie à la transition t, c-à-d :*

$$Pre(p,t) \leq M(p);$$

Note : t est franchi par :

$$Pre(.,t) \leq M;$$
$$M(t >.$$

Le franchissement d'une transition permet d'atteindre un nouveau marquage M' tel que :

$$\forall p \in P, M' = M + C \tag{1.3}$$

Donc :

$$M(t > M'$$

Exemple

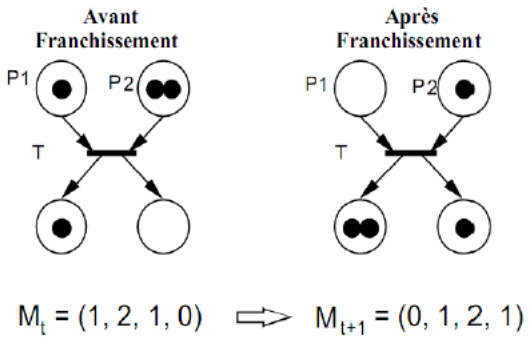

FIGURE 1.2 – Processus du Franchissement d'un *RdP*

La figure précédente illustre le passage et le processus du franchissement. Au départ, le marquage était $M_t = (M(p_1), M(p_2), M(p_3), M(p_4)) = (1, 2, 1, 0)$, mais après l'activation de t, ainsi, on obtient $M_{t+1} = (0, 1, 2, 1)$. En enlevant donc un jeton de chaque place dans un premier temps, ce changement est effectué. M_{t+1} n'est donc plus considéré comme franchissable car P_1 est vide (ne contient pas de jetons).

Définition 1.3.2 (Séquences de Franchissement). *On considère (R, M_0) un RdP marqué et on a $s = \{t_i \in T^* , i \in [1, n]\}$ la séquence de transition. s est dite franchissable à partir de M ssi il existe un ensemble de marquages $\{M_i, i \in [1, n]\}$ vérifiants :*

$$M \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} M_n \tag{1.4}$$

On peut remarquer que le sous ensemble T^* de T inclut des transitions qui constituent la séquence de franchissement.
Donc, le tir s résulte au fait qu'on marque finalement M_n, par conséquent, on note $M(\ s > M_n)$.

Définition 1.3.3 (Marquage atteignable). *Un marquage M est atteignable, dit aussi accessible, à partir d'un marquage initial M_0 est un marquage pour lequel il existe une séquence de franchissement qui, à partir de M_0 aboutisse à M c-à-d : $s \in T^*$ et $M_0(\ s > M$.*

Définition 1.3.4 (Ensemble des marquages accessibles). *L'ensemble d'accessibilité (où ensemble des marquages accessibles) d'un RdP (R, M_0) est l'ensemble de tous les marquages qui peuvent être atteints à partir du marquage initial en suivant les transitions du réseau, noté par : $A(\ R, M_0)$ tel que :*

$$A(R, M_0) = \left\{M \in \mathbb{N}^P / \exists s \in T^* \ telque \ M_0(\ s > M\right\} \tag{1.5}$$

Définition 1.3.5 (Graphe des marquages atteignables). *On peut représenter l'ensemble de marquage accessible par un graphe si ce dernier est fini. Le graphe de marquage a comme sommet l'ensemble $A(R, M_0)$. Un arc relie deux sommet M et M' s'il existe une transition t franchissable qui permet le passage d'un marquage à un autre $M(\ t > M'$. Les arcs du graphe sont étiquetés par les transitions correspondantes.*

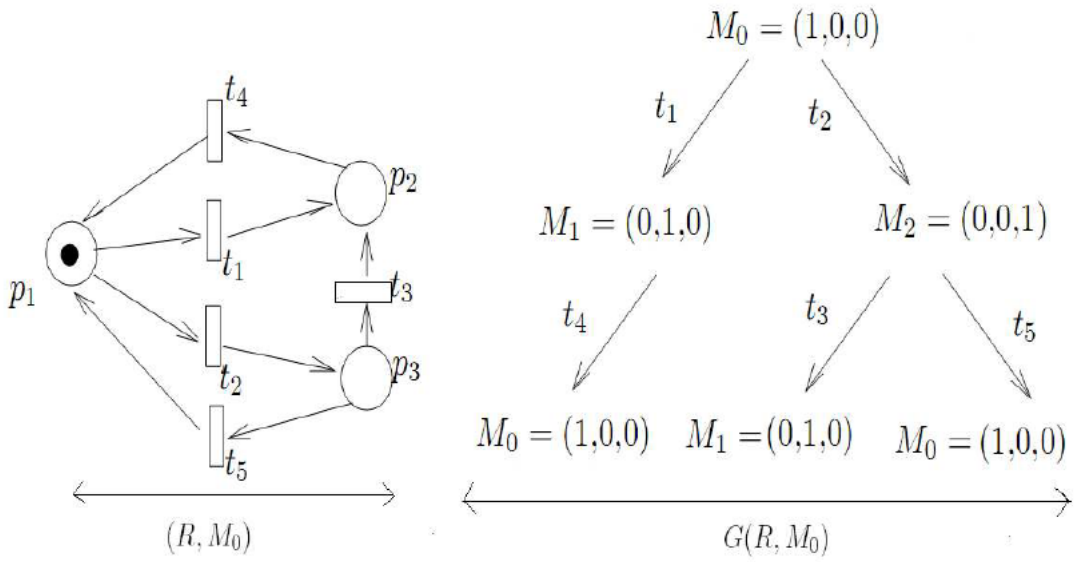

FIGURE 1.3 – Illustration d'un graphe de marquages atteignable

1.3.2 Conflit et Transition Parallèle

L'évolution d'un réseau de Petri se réalise par le franchissement de transitions sensibilisées, une seule à la fois. Si les transitions ne partagent pas de place, il n'y a pas de conflit. Sinon, des transitions en conflit peuvent provoquer un comportement imprévisible en raison de la compétition pour les ressources.

Définition 1.3.6. *Un conflit structurel correspond à l'existence d'une place p_1 qui a au moins deux transitions de sortie t_1, t_2, i.e :*

$$\exists p \in P, Pre(\, p, t_1) \times Pre(\, p, t_2) \not\equiv 0 \tag{1.6}$$

Elle sont en conflit effectif pour un marquage M si de plus :

$$M(\, t_1 >; \tag{1.7}$$

et

$$M(\, t_2 >; \tag{1.8}$$

et

$$\exists\, p \in P : M(p) < Pre(p, t_2) + Pre(p, t_1) \tag{1.9}$$

Définition 1.3.7 (Parallèlisme). *Deux transitions parallèles dans un réseau de Petri sont des transitions qui peuvent se produire simultanément et indépendamment l'une de l'autre.*

1.3.3 Caractérestiques d'un réseau de Petri

Le RdP fournit diverses méthodes pour l'analyse les des propriétés permettant de mesurer la qualité du système voulu. Ci-dessous nous présentons les propriétés les plus importantes des réseaux de Petri.

a) **Réseau borné**
La bornitude d'un réseau de Petri indique que ce dernier peut être modélisé par un nombre d'états possibles limité, impliquant que le nombre de marquages accessibles est également fini. En revanche, si le réseau de Petri est non borné, cela indique l'infinité du nombre d'états, ce qui est expliqué par la présence de certains paramètres non bornés du système.

Définition 1.3.8. *Soit un RdP, une place $p \in P$ est dite k-bornée pour un marquage intial M_0 ssi :*

$$\exists\, k \in \mathbb{N}, \forall M' \in A(R, M_0), M'(p) \leq k; \tag{1.10}$$

avec :
• $k \in \mathbb{N}$.
• La place p est dite sauve si $k = 1$.

Définition 1.3.9. *Un RdP est dit k-borné pour un marquage initial M_0, ssi toutes ses places sont k-bornée.*

Définition 1.3.10. *Un Rdp marqué (R, M_0) est dit borné, si les places contiennent un nombre de jetons.*

b) **Réseau sans blocage**
Un réseau de Petri est qualifié de bloqué lorsqu'à un moment donné, son processus d'avancement cesse et aucune transition n'est possible. On peut définir ça formalemment comme suit :

Définition 1.3.11. *On appelle marquage puit (bloqué), chaque marquage M' d'un RdP (R, M_0) pour leuqel, quelque soit la transition t, t n'est pas franchissable à partir de M'.*

Un Rdp est sans blocage pour un marquage M_0 initial, si tout marquage est atteignable depuis M_0.

c) **Vivacité**

Une caractéristique essentielle afin de garantir un fonctionnement optimal d'un système. Le réseau résultant, dit vivant, représente un système qui fonctionne sans aucun blocage, en continu.

Définition 1.3.12. *Un RdP est vivant pour un marquage initial si toutes ses transitions sont vivantes pour M_0 :*

$$\forall M \in A(R, M_0), \forall t \in T, \exists M' \in A(R, M)\, telque\, M'(t > \tag{1.11}$$

Dit autement,

$$\forall M \in A(R, M_0), \forall t \in T, \exists s \in T^*\, telque\, M(st > \tag{1.12}$$

d) **Etat d'accueil**

Un état d'accueil signifie que le système peut arriver à cet état à partir des marquages précédents. Si l'état initial est un état d'accueil, cela veut dire que le système peut toujours être remis à zéro. Cette idée se formalise dans la formulation qui suit :

Définition 1.3.13. *un RdP dispose d'un état d'accueil pour un marquage initial M_0 si :*

$$\forall M' \in A(R, M_0), \exists s \in T^*/M'(s > M \tag{1.13}$$

Remarque 1.3.1. *Le système représenté par un RdP peut être réinitialisé si le marquage initial constitue un état d'accueil.*

1.4 Avancée dans les réseaux de Petri

Plusieurs extensions ont été développées pour les réseaux de Petri classiques, conduisant à la création de plusieurs formalismes de réseaux de Petri. Ces formalismes intègrent notamment des aspects temporels, stochastiques, ce qui a apporté une plus grande richesse aux structures des RdP.

1.4.1 Réseaux de Petri généralisés (RdPG)

Le système représenté par un RdP peut être réinitialisé, si le marquage initial constitue un état d'accueil.
L'arc de P_i à T_j a un poids. La transition T_j ne sera validée que si P_i contient au moins p jetons. Lors du franchissement de cette transition, p jetons seront retirés de la place P_i. Lorsqu'un arc T_j à P_i a un poids p cela signifie que lors du franchissement de T_J, p jetons seront ajoutés a la place P_i.

Exemple

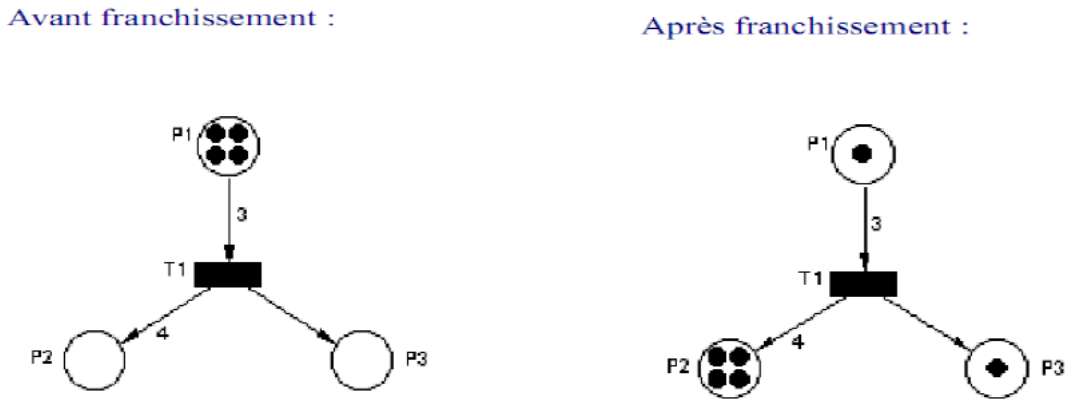

FIGURE 1.4 – Illustration d'un *RdPG*

1.4.2 Réseaux de Petri inhibiteurs

Les réseaux de Petri à arc inhibiteur (ou réseaux de Petri inhibiteurs) sont une variante des réseaux de Petri classiques. En plus des éléments traditionnels (places, transitions et arcs), ils introduisent des arcs inhibiteurs qui permettent de modéliser des conditions de non-présence de jetons dans une place. La définition formelle des RdP avec arcs inhibiteurs est la suivante :

Définition 1.4.1. *Un réseau de Petri à arcs inhibiteur est défini par un 5-uplet* $R = (P, T, Pre, Post, Inh)$ *ou :*
• $P = \{p_i, i \in [1, n]\}$*, ensemble de places fini et non vide.*
• $T = \{t_i, i \in [1, m]\}$*, ensemble de transitions fini et non vide.*
•$Post$ *et* $Pre : T \times P \longrightarrow \mathbb{N}^*$ *respectivement les fonctions d'incidence arrière et d'incidence avant ;*
•$Inh : P \times T \longrightarrow \mathbb{N}^*$ *la fonction d'inhibition.*

La définition suivante explique comment une transition se produit dans un RdP avec un arc inhibiteur.

Définition 1.4.2. *Soit* M *un marquage d'un réseau de Petri à arcs inhibiteurs et* t *une transition : la transition* t *est franchissable à partir de* M *si et seulement si :*

$$\forall p \in P, M(p) \geq Pre(p, t) \, et \, M(p) < Inh(p, t) \tag{1.14}$$

Le franchissement de t *à partir de* M *conduit au marquage* M_0 *défini par :*

$$\forall p \in P, M'(p) = M(p) - Pre(p, t) + Post(p, t) \tag{1.15}$$

Dans les réseaux de Petri inhibiteurs, seule la condition de franchissement change. Visuellement, ces réseaux se différencient des réseaux standard par une flèche avec un petit cercle à l'extrémité pour indiquer un arc inhibiteur, au lieu d'une flèche simple.

FIGURE 1.5 – Arc Orienté vs Arc Inhibiteur

1.4.3 Réseaux de Petri colorés

Afin d'augmenter la richesse de l'information véhiculée par les jetons, les RdP colorés ont été introduit, ce qui les rend particulièrement utiles pour la modélisation de systèmes de grande envergure et complexes.

• Pour différencier les jetons, on leur attribue des couleurs spécifiques. Chaque emplacement est alors associé à un ensemble de couleurs représentant les jetons pouvant s'y trouver. De même, chaque transition est associée à un ensemble de couleurs correspondant aux façons dont la transition peut être effectuée.
• Les transitions dans un RdP coloré peuvent franchir les jetons de différentes manières en fonction des couleurs qui leur sont attribuées. Cette relation entre les couleurs de franchissement et le marquage coloré est déterminée par des fonctions associées aux arcs.
• Dans un réseau de Petri coloré, la couleur peut être un ensemble de valeurs, ce qui permet de représenter des informations complexes comme la nature et la position d'un objet, ou même une trame de données comprenant à la fois une valeur et une adresse de destination. En combinant les concepts précédemment introduits, on peut définir un réseau de Petri coloré comme suit :

Définition 1.4.3. *Un RdP coloré est défini par un ensemble de sept éléments, à savoir :*

$$R_c = (P, T, A, C_{coul}, W, I, M_0)$$

• $P = \{p_i, i \in [1, n]\}$*, ensemble de places fini et non vide.*
• $T = \{t_i, i \in [1, m]\}$*, ensemble de transitions fini et non vide.*
• A *est un ensemble fini d'arcs.*
•C_{coul} *est un ensemble fini de couleurs ou type de données.*
• W *est une fonction de poids des arcs :*

$$W : A \to \mathbb{N} \times C$$

Elle associe à chaque arc un poids (nombre naturel) et un type de jeton (couleur).
• I *est la fonction d'incidence* $C = Post - Pre$;
• M_0 *est le marquage initial :*

$$M_0 : P \to \mathbb{N}$$

Remarque 1.4.1. *Les RdP colorés ne sont pas plus puissants comme les RdP classiques ; ils offrent une manière plus concise pour les représenter. La transformation d'un RdP ordinaire en un réseau de Petri coloré est appelée "pliage", tandis que le "dépliage" désigne la transformation inverse.*

1.4.4 Réseaux de Petri temporisés

Un RdP temporisé constitue une extension d'un RdP ordinaire. Cette dernière se distingue par l'incorporation de temporisations, c'est-à-dire l'introduction de la notion de temps. On distingue deux types de réseaux de Petri temporisés : les réseaux de Petri T-temporisés [14] et les RdP P-temporisés [15].

A) Réseau de Petri P-temporisé

Définition 1.4.4. *Un réseau de Petri P-temporisé est défini par le couple (R, d) avec :*

1. *R : un RdP.*
2. *la fonction $d : T \longrightarrow Q^+$ désigne la temporisation dans un réseau de Petri. Cette fonction joue un rôle essentiel dans la règle de franchissement, qui doit prendre en compte l'écoulement d'un certain temps. Lorsqu'une marque est placée dans une place p à l'instant t_0, elle demeure indisponible (gelée) pour une durée précisée par $d(p)$, soit dans l'intervalle $[t_0, t_0 + d(p)]$. Passé ce délai, la marque devient disponible pour déclencher le franchissement d'une transition.*

Les RdP temporisés sont fréquemment employés pour formaliser les synchronisations dans les systèmes. L'exemple suivant illustre un RdP P-temporisé.

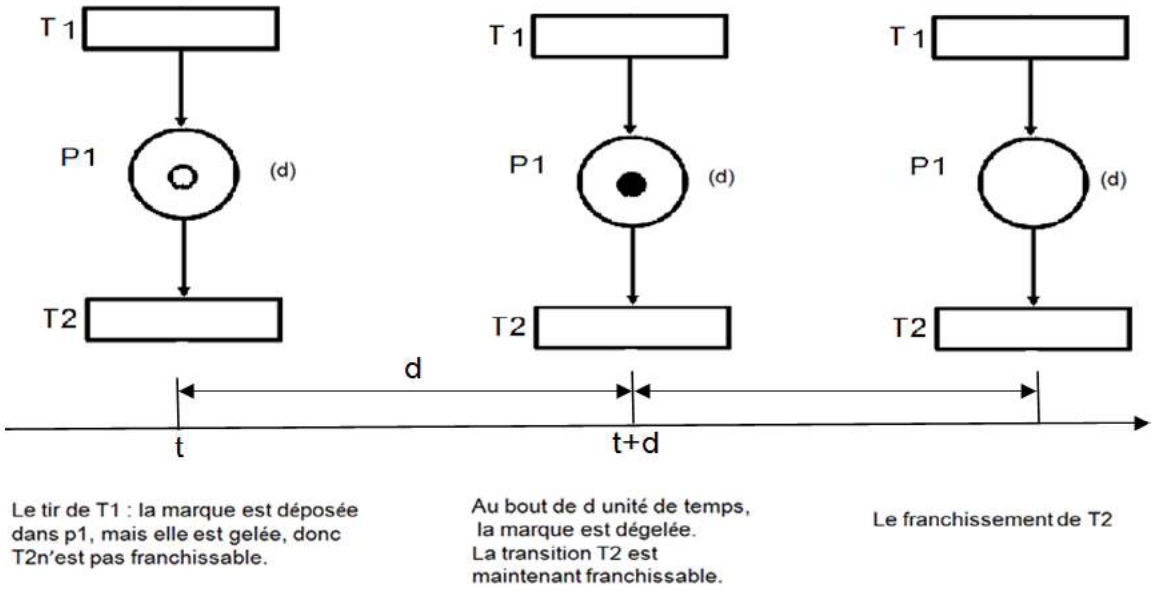

FIGURE 1.6 – Illustration d'un RdP Temporisé

B) Réseaux de Petri T-temporisés
Une durée d'activation pour les transitions est la durée pendant laquelle un jeton situé dans chaque place amont de la transition activée est « réservé » pour cette transition (avant de disparaître), et au delà de laquelle un jeton apparaît dans chacune des places en aval.
Dans cette illustration, la transition T_2 se produit de manière continue et sans interruption.

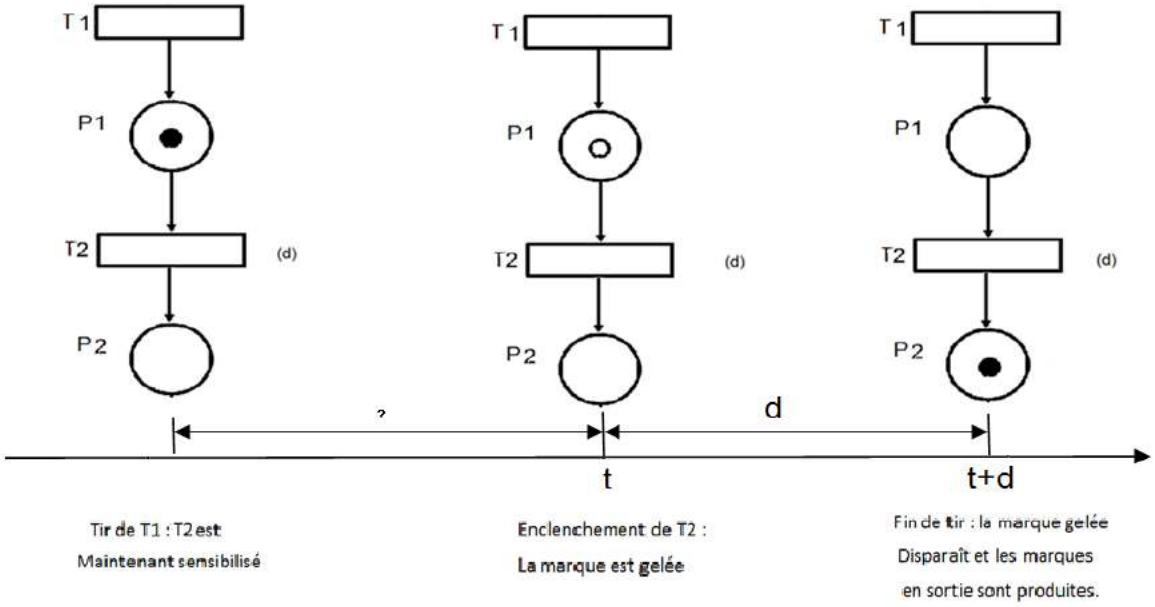

FIGURE 1.7 – *RdP* avec des Transitions Temporisées

Après sensibilisation de la transition T_2 la marque est indisponible. Il est nécessaire d'attendre une unité de temps avant que le jeton n'apparaisse dans P_2. Durant cette période, ce jeton ne peut plus être employé pour la validation des autres transitions.

1.4.5 Réseaux de Petri temporels

Les réseaux de Petri temporels (ou réseaux de Petri temporisés) sont une extension des réseaux de Petri classiques qui intègrent des notions de temps dans la modélisation. Ils permettent de représenter et d'analyser des systèmes où le facteur temps est crucial, comme les systèmes temps réel, les protocoles de communication, ou encore les systèmes de production.

Définition 1.4.5. *Un RdP temporel est une paire $N_t = (R, I_s)$ où*

1. *R est un réseau de petri marqué*
2. *I_s est la fonction durée de franchissement :*

$$I_s : T \longrightarrow (\mathbb{Q}^+ \cup \infty)^2$$
$$t_i \longrightarrow I_{s_i}(t_i) = [a_i, b_i].$$

Chaque transition t_i du réseau est associée à un intervalle de temps défini par $I_{s_i} = [a_i, b_i]$, où a_i correspond à sa date de tir au plus tôt et b_i à sa date de tir au plus tard.

1.4.6 Réseaux de Petri stochastique

Les réseaux de Petri stochastiques ont été pionniers grâce aux travaux de Natkin et Molloy [12, 10], qui ont introduit cette extension novatrice des réseaux de Petri. Depuis lors, de nombreux modèles ont été développés, et des synthèses exhaustives ont été proposées dans [2] pour aborder les défis d'évaluation quantitative des systèmes informatiques industriels.

Dans les RdP stochastiques, les transitions sont associées à des délais aléatoires, contrairement aux durées déterministes et constantes des réseaux de Petri temporisés. Ces délais sont modélisés par des variables aléatoires, souvent considérées comme étant exponentielles.
Cette approche permet de représenter les graphes des marquages par un processus appelé markovien.

Les RdP temporels exploitent un intervalle complet de valeurs $[\theta_{\min}, \theta_{\max}]$, tandis que dans les RdPS, la durée de sensibilisation θ est une variable aléatoire soumise à une distribution de probabilité spécifique, comme celle de la distribution exponentielle :

$$P_\theta(x) = P[\theta \leq x] = 1 - e^{-\lambda x}$$

La fonction $P_\theta(x)$ indique la probabilité que le franchissement se produise avant x, ce qui signifie que la durée de sensibilisation est inférieure ou égale à x.
À partir de cela, on peut déterminer la valeur moyenne de la durée de sensibilisation :

$$\bar{\theta} = \int_0^\infty (1 - P_\theta(x))\,\mathrm{d}x = \int_0^\infty e^{-\lambda x}\,\mathrm{d}x = \frac{1}{\lambda};$$

Le taux de franchissement d'une transition est défini par (λ >0).

Grâce à la propriété sans mémoire des distributions exponentielles pour les délais de franchissement, Molloy [11] a démontré que les réseaux de Petri stochastiques sont isomorphes aux processus de Markov à temps continu avec un espace d'états discret. En particulier, un réseau de Petri stochastique k-borné est équivalent à un processus de Markov à espace d'états fini, permettant ainsi l'application des techniques propres aux processus markoviens pour calculer les mesures de performances.

Les réseaux de Petri stochastiques présentent plusieurs avantages notables : leur grande flexibilité, leur capacité équivalente à celle des processus de Markov en termes de modélisation, ainsi qu'une représentation graphique intuitive.

La validation des propriétés structurelles des réseaux de Petri est essentielle pour certifier les modèles associés. Bien que la construction de modèles de réseaux de Petri stochastiques pour des systèmes réels soit initialement complexe et nécessite un travail considérable, une fois les modèles établis, les possibilités d'analyse qu'ils offrent justifient pleinement ces efforts.

Avec les Réseaux de Petri Stochastiques (RdPS), il sera possible de calculer diverses métriques telles que le temps de bon fonctionnement entre deux pannes, le temps de réparation, ou encore, dans certains cas, la durée opérationnelle d'une machine, les taux de production, l'évolution des stocks, etc. La modélisation des systèmes nécessite parfois de considérer des transitions non stochastiques (immédiates). Pour répondre à ce besoin, une extension des réseaux de Petri stochastiques a été proposée, appelée "Réseaux de Petri Stochastiques Généralisés" (RdPSG) .

Nous proposons une méthode d'évaluation des performances des files d'attente avec priorité en utilisant les réseaux de Petri stochastiques. Nous avons opté pour les RdPSG pour évaluer les performances des systèmes avec priorité et sources finies. Ce choix découle des avantages offerts par les RdPSG, notamment :

— leur capacité de modélisation avancée, notamment leur capacité à intégrer différentes formes de dépendances qui déterminent le fonctionnement d'un système : la synchronisation, la concurrence et le parallélisme,

— leur capacité à analyser les propriétés du système et à valider le modèle établi,

— leur capacité à évaluer les mesures du comportement du système, notamment les indices de performances.

Suite à ces avantages les RdPSG ont été explicités pour l'analyse des performances des systèmes de files d'attente compliqués.

1.5 Réseaux de Petri Stochastique Généralisés(RdPSG)

Les RdP stochastiques généralisés étendent les capacités des RdP stochastiques en incorporant deux types de transitions distincts :

• Les transitions immédiates, qui sont franchies instantanément dès leur activation, sans aucune temporisation.

• Les transitions temporelles, associées à des variables aléatoires qui déterminent la durée nécessaire pour leur franchissement.

Les taux de transition peuvent varier en fonction du marquage. En cas de plusieurs transitions franchissables simultanément, les transitions immédiates ont la priorité sur les transitions temporelles et sont donc traitées en premier. Cela permet leur franchissement dès leur activation. Lorsque plusieurs transitions immédiates sont activées simultanément, une distribution de probabilité doit spécifier laquelle d'entre elles sera franchie en premier.

Les transitions immédiates représentent des actions du système se terminant instantanément par rapport aux autres durées du système. Les états qui ne permettent que des transitions temporelles sont appelés états tangibles. En revanche, les états avec au moins une transition immédiate activée sont qualifiés d'états évanescent.

En résumé, un réseau de Petri stochastique généralisé distingue deux types d'états :

• Les états tangibles, où toutes les transitions activées sont temporelles.

• Les états évanescent, caractérisés par au moins une transition immédiate activée.

Remarque 1.5.1. *Dans la représentation graphique d'un RdPSG, les transitions immédiates sont représentées par des traits, tandis que les transitions temporelles sont symbolisées par des rectangles.*

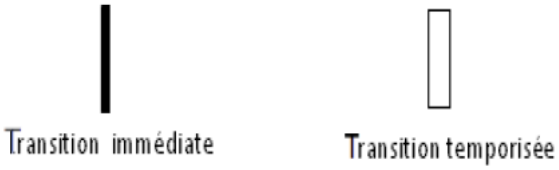

FIGURE 1.8 – Transition immédiate *VS* Transition Temporisée

Définition 1.5.1. *Un réseau de Petri stochastique généralisé est un 8-uplet* $(P, T, Pre, post, Inh, pri, W, M_0)$ *où :*

- $P = \{p_i, i \in [1, n]\}$*, ensemble de places,*
- $T = \{t_i, i \in [1, m]\}$*, ensemble de transitions temporisées et des tranisitons immédiates,*
- $Post, Pre, Inh : P \times T \longrightarrow \mathbb{N}$ *sont les fonctions d'incidence arrière, d'incidence avant et d'inhibition respectivement ;*
- $pri : T \longrightarrow [0, 1]$ *est la fonction de priorité qui associe à chaque transition temporisée la valeur 0 et à chaque transition immédiate la valeur 1 ;*
- $W : T \longrightarrow \mathbb{R}^+$ *est une fonction qui associe à chaque transition temporisée un taux de franchissement ;*
- $M_0 : P \longrightarrow \mathbb{N}$ *est le marquage initial du réseau.*

Ce formalisme spécifique des RdPSG pose une question fondamentale : est-ce que l'introduction des transitions immédiates altère l'homogénéité markovienne du processus de marquage ?

Selon Ajmone et ses collaborateurs, il a été démontré que si aucune séquence infinie d'états évanescent accessibles de manière successive n'existe, alors le processus de marquage maintient son caractère markovien homogène après l'élimination des états évanescents.

1.6 Conclusion

Le formalisme des réseaux de Petri, conçu pour résoudre les défis liés à la concurrence, la synchronisation et le parallélisme, constitue un outil indispensable pour la spécification fonctionnelle d'un problème tout en mettant en évidence ses contraintes. Les propriétés mathématiques découlant de l'analyse des réseaux de Petri facilitent une étude détaillée du comportement et de la structure, jouant un rôle crucial dans la validation des spécifications.

Dans ce chapitre, nous avons introduit le modèle fondamental des réseaux de Petri, en mettant l'accent sur leur dynamique et leur comportement. Nous avons également exploré différentes extensions apportées à ces modèles de base. Parmi celles-ci, les Réseaux de Petri Stochastiques Généralisés (RdPSG) seront utilisés dans les chapitres à venir pour l'analyse des systèmes à priorités.

Chapitre 2

Les Réseaux Peer-to-Peer

2.1 Introduction

Ces dernières années, les réseaux ont connu une croissance significative en raison des avantages qu'ils offrent, tels que l'échange d'informations et le partage de ressources. Les réseaux se basent sur l'architecture cliant/serveur.

L'architecture peer-to-peer ($P2P$) propose une solution différente à celle de l'architecture client-serveur, facilitant la répartition du trafic et de la charge, améliorant la résistance aux pannes et offrant un degré d'anonymat.

Nous allons montrer dans ce chapitre une vue d'ensemble des réseaux des réseaux peer-to-peer ($P2P$), abordant leur définition, une comparaison avec l'architecture client-serveur, leurs propriétés, leur catégories, leurs avantages et inconvénients, ainsi que leur domaine d'application.

2.2 Définition

Lors de la revue littérature pour ce mémoire, nous avons trouvés plusieurs définitions du réseau $P2P$. Parmi ces définitions nous citons ci-dessous une des définition que nous avons découvertes dans [9] :

Le terme "Peer-to-Peer" ($P2P$), également connu sous le nom de "poste à poste" ou "pair à pair" [16]en français, désigne un modèle de réseau informatique dans lequel les éléments (les nœuds ou "pairs") agissent à la fois en tant que clients et serveurs lors des échanges.

Les réseaux Peer-to-Peer, comme leur nom l'indique, facilitent les communications directes, de pair à pair, entre les différents nœuds du réseau. Cela permet aux nœuds d'échanger divers types d'informations sans nécessiter un serveur central.

L'utilisation des réseaux $P2P$ connaît actuellement une forte expansion en raison de leurs avantages, ainsi que de leur expansion dans tous les domaines. En conséquence, de nombreux logiciels ont été développés pour répondre à cette demande croissante.

2.3 Comparaison entre Client / Serveur et $P2P$

La technique client/serveur permet l'échange de services entre ordinateurs.
Dans cette architecture, il y a une unique entité centrale extrêmement puissante, le serveur, et plu-

sieurs entités généralement moins puissantes, les clients. Le serveur est le seul fournisseur de services aux clients [7].

L'architecture $P2P$ est une alternative à l'architecture client/serveur, offrant de nombreux avantages par rapport à d'autres architectures . Ce tableau montre la différence entre les deux architectures.

Critère	**Client/serveur**	**Modèle** $P2P$
Gestion	Supervisé	Auto-organisé
Présence	Permanante	Ah-doc
Accès aux ressources	Recherche	Découverte
Organisation	Héirarchique	Dsitribué
Mobilité	Statique	Mobile
Disponibilité	Dépendante du serveur	Indépendante des pairs

TABLE 2.1 – Infrastructures Client-serveur *VS* Modèle $P2P$

2.4 Propriétés du $P2P$

Dans cette section, nous décrivons les caractéristiques principales du modèle Peer-to-Peer :

•**Décentralisation :** où chaque nœud du réseau peut agir en tant que client et serveur, ce qui évite les goulets d'étranglement.

•**Passage à l'echelle :** L'objectif est de permettre la coopération entre un grand nombre de nœuds (pouvant aller jusqu'à des milliers voire des millions) pour partager leurs ressources tout en maintenant de bonnes performances du système. Cela implique qu'un système $P2P$ doit proposer des méthodes bien adaptées à un environnement où il y a un grand volume de données à partager, un nombre important de messages à échanger entre de nombreux nœuds partageant leurs ressources via un réseau largement distribué.

•**L'auto-organisation :** Comme les systèmes $P2P$ sont souvent déployés sur Internet, l'ajout d'un nouveau nœud à un système $P2P$ ne nécessite pas une infrastructure coûteuse. Il suffit d'avoir un accès à Internet et de connaître un autre nœud déjà connecté pour rejoindre le système. Un système $P2P$ doit être un environnement ouvert, ce qui signifie qu'un utilisateur sur un nœud doit pouvoir connecter son nœud au système sans contacter une personne ni passer par une autorité centrale.

•**Autonomie des nœuds :** Chaque nœud gère ses ressources de manière autonome, décidant quelle partie de ses données à partager. Il peut se connecter et se déconnecter à tout moment. De plus, il a le contrôle total sur sa puissance de calcul et sa capacité de stockage.

• **Scalabilité :** Les réseaux pair-à-pair peuvent facilement s'adapter à une grande quantité de nœuds. En ajoutant simplement de nouveaux nœuds au réseau, sa capacité et sa performance peuvent être augmentées.

• **Résilience :** En raison de leur nature décentralisée, les réseaux pair-à-pair sont souvent plus résilients face aux pannes. Si un nœud échoue, les autres nœuds peuvent continuer à fonctionner normalement.

2.5 Catégories des réseaux $P2P$

Il existe trois principales catégories de systèmes $P2P$ peuvent être distinguées, comme le montre la figure ci-dessous : centralisé, décentralisé et hybride. La catégorie décentralisée est divisée en décentralisé structuré et décentralisé non structuré.

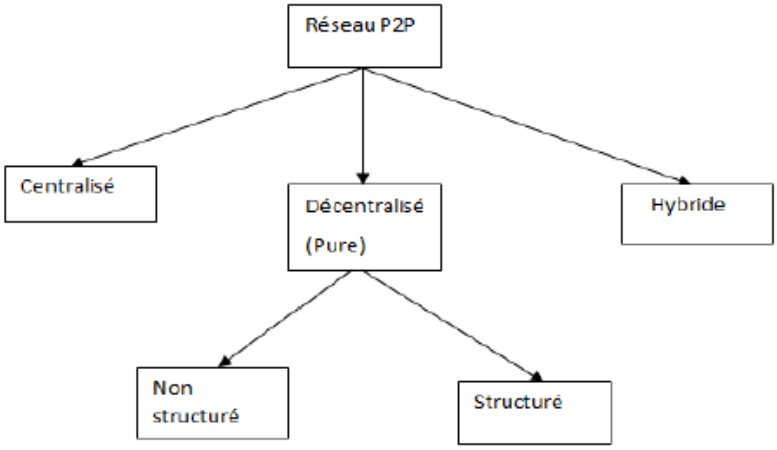

FIGURE 2.1 – Catégories des réseaux $P2P$

2.5.1 Architecture centralisée

Cette architecture représente la première génération des réseaux $P2P$. Elle repose sur un serveur central contenant des métadonnées décrivant les ressources (notamment des fichiers) partagées par les participants du réseau. Les utilisateurs se connectent à ce serveur pour communiquer entre eux et échanger des fichiers.

Le serveur joue un rôle principal dans la gestion de la recherche des ressources en identifiant les nœuds qui stockent les fichiers. Une fois connecté au serveur et après avoir découvert le nœud possédant le fichier recherché, l'échange et la communication de ce fichier entre le pair demandeur se font directement, cette figure montre cette architecture :

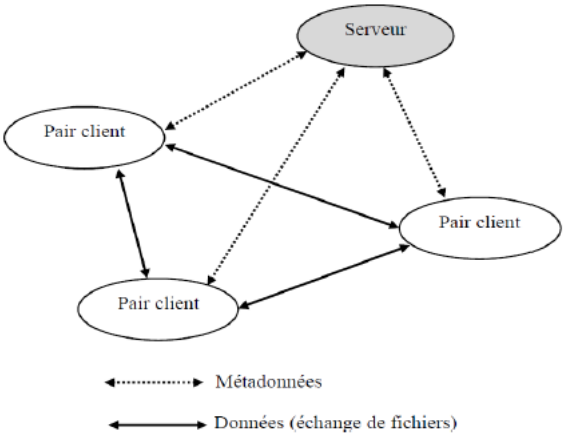

FIGURE 2.2 – Modèle centralisé

Dans cette architecture (comme dans l'exemple de Napster), chaque pair héberge un ensemble de fichiers qu'il partage avec les autres pairs du réseau. Pendant ce temps, le serveur sauvegarde deux types de données :

- Une table contenant des informations sur les pairs connectés, telles que leur adresse IP, numéro de port, bande passante disponible, etc.
- Une table répertoriant tous les fichiers partagés par chaque pair, avec des métadonnées décrivant chaque fichier (nom, date de création, etc.).
  Les étapes suivies par un utilisateur sont les suivantes :
- L'utilisateur se connecte au serveur central pour annoncer les fichiers qu'il souhaite partager,
- Lorsqu'il recherche un fichier, l'utilisateur envoie une requête au serveur,
- Le serveur recherche dans sa table d'index et renvoie la liste des pairs qui possèdent le fichier recherché,
- Enfin, le client pair établit une connexion directe avec un ou plusieurs pairs hébergeant le fichier recherché, et le télécharge directement sans passer par le serveur.

Napster est le premier réseau $P2P$ conçu pour le partage de fichiers, en particulier pour l'échange de fichiers audio/vidéo.

2.5.2 Architecture décentralisée

Cette architecture est de nature plate, où tous les nœuds du réseau remplissent les mêmes fonctions. Ils agissent à la fois en tant que serveurs et clients. Les pairs dans un tel réseau sont souvent appelés des "servents" (SERVeur+cliENT). Étant donné qu'il n'y a pas de serveur central, le départ d'un pair n'affecte pas le système. Deux types d'architectures décentralisées peuvent être distingués : non structuré et structuré.

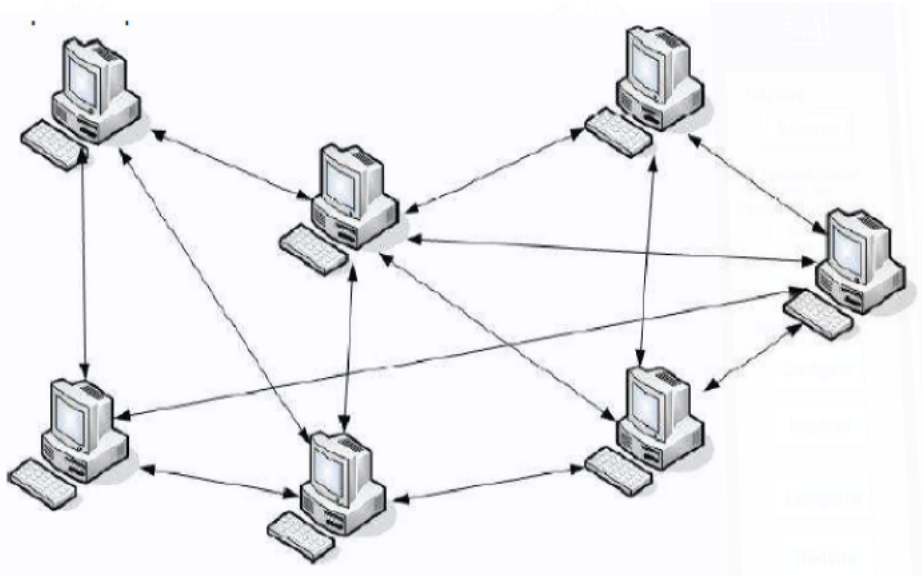

FIGURE 2.3 – Modèle décentralisée

Architecture non-structurée :

Cette classification est basée sur le concept d'inondation. Lorsqu'un utilisateur découvre une ressource, il envoie une requête contenant le nom de la ressource à ses voisins, jusqu'à ce qu'elle atteigne le client qui possède l'objet recherché. Pour éviter que le réseau ne soit inondé pendant trop

longtemps, le système associe à chaque requête un temporisateur TTL (Time To Live), généralement avec une valeur de 7. Lorsque le TTL atteint zéro, la requête n'est plus transmise.

Le principal inconvénient de ce mécanisme est que si le TTL atteint 0 avant d'avoir parcouru l'intégralité du réseau, cela peut entraîner l'échec d'une recherche même si l'objet recherché est disponible sur le réseau $P2P$.

Ces types d'architectures offrent une grande résistance aux entrées et sorties de nœuds du système. Cependant, le mécanisme de recherche actuel ne s'adapte pas efficacement à l'échelle, ce qui entraîne une charge importante pour les participants du réseau. Ces approches sont mises en œuvre dans des systèmes tels que Gnutella et FreeNet.

Nous présentons ici un exemple de réseau $P2P$ décentralisé et non structuré, tel que Gnutella, que nous avons découvert dans [13].

• Gnutella :

Gnutella a été pionnier en tant que premier réseau P2P entièrement décentralisé. Lancé en mars 2000 par Justin Frankelet et Tom Pepper en réponse aux limitations de centralisation de Napster, Gnutella a su tirer parti de cette expérience. Ce protocole ouvert et décentralisé permet des recherches distribuées sur une topologie plate de pairs.

D'après Gnutella, chaque nœud du réseau agit à la fois en tant que serveur et client. Ce protocole ne repose pas sur un répertoire centralisé et ne contrôle ni la topologie ni l'emplacement des fichiers. Le réseau se forme lorsque les nœuds rejoignent le réseau en suivant des règles simples. L'emplacement des données n'est pas basé sur une connaissance de la topologie. Pour localiser un objet, un client interroge ses voisins qui, à leur tour, interrogent leurs propres voisins. Ce système permet l'entrée et la sortie des clients, mais son mécanisme ne s'adapte pas bien à une grande échelle et génère des charges élevées dans le réseau.

Architecture structurée

Les systèmes P2P décentralisés structurés utilisent un algorithme de recherche entièrement déterministe, où les connexions entre les pairs sont établies selon des règles bien définies.

Une architecture structurée dans le contexte des réseaux pair-à-pair ($P2P$) est un modèle où les pairs sont organisés de manière systématique et prévisible, souvent à l'aide de tables de hachage distribuées (DHT) ou d'autres structures de données efficaces. Ce type d'architecture permet une recherche de ressources plus rapide et plus fiable, car chaque pair a une connaissance partielle mais précise de l'emplacement des ressources dans le réseau. Les exemples typiques d'architectures structurées incluent Chord, Kademlia et Pastry, qui utilisent des algorithmes sophistiqués pour assurer une distribution équilibrée et une récupération rapide des données.

Les DHT offrent des avantages significatifs en raison de leur modèle P2P pur : aucun nœud ne joue un rôle central ou particulier, et tous agissent de manière équivalente. Les propriétés des DHT sont :
• **Performance :** En situation normale, le nombre de sauts nécessaire est limité.
• **Passage a l'échelle (évoluvité) :** Les DHT bénéficient de deux caractéristiques qui leur permettent de bien passer à l'échelle. Premièrement, le nombre moyen de sauts nécessaires pour router les requêtes reste faible, même dans les communautés comptant un grand nombre de participants. Deuxièmement, les tables de routage restent d'une taille raisonnable par rapport au nombre de participants.
• **Fiabilité :** En utilisant un algorithme de découverte et de routage, il est possible de déterminer le pair dont l'identifiant est le plus proche pour une clé donnée. Dans des conditions statiques, une

réponse négative à une requête indique que la ressource demandée n'est pas disponible dans la communauté.
• **Tolérance aux fautes :** En raison de leur nature décentralisée, qui exclut tout point central, les DHT présentent une bonne tolérance aux suppressions aléatoires de nœuds. Les requêtes peuvent être acheminées même si certains nœuds disparaissent. Cependant, chaque nœud racine d'une ressource particulière peut être considéré comme un point central. Des mécanismes de redondance sont souvent mis en place pour éviter l'inaccessibilité d'une ressource présente dans une DHT. Un exemple de réseau P2P décentralisé et structuré est Chord.

***Chord**

Chord structure son espace d'adressage en suivant un anneau où les 2^m adresses (ou identifiants, id) possibles sont ordonnées le long de sa circonférence. Chaque pair ainsi que chaque ressource possède un identifiant obtenu par une fonction de hachage SHA-1 (dans ce cas, m = 160 bits), assurant ainsi une répartition homogène des ressources sur l'anneau Chord.

Chord se base sur une fonction de routage simple, où étant donné un identifiant, Chord localise le pair responsable qui possède le plus petit identifiant supérieur ou égal à celui de la ressource. Lorsqu'un pair rejoint le réseau, il prend en charge une partie des identifiants attribués à son successeur direct, et lorsqu'il quitte le réseau, tous ses identifiants sont attribués à son successeur. Chord ne propose donc pas toutes les primitives nécessaires pour stocker, rechercher et répliquer les données dans une DHT, et encore moins pour exécuter une application de partage de fichiers fonctionnelle.

L'intérêt de Chord réside dans sa capacité à offrir une structure simple et efficace pour router les messages. À cet effet, chaque nœud du réseau maintient une table de pointeurs contenant les informations (id, ip, port) pour un certain nombre de pairs. Plus cette liste est longue, plus elle est coûteuse à maintenir, mais plus le routage est rapide (dans le cas extrême où l'on connaît tous les pairs du réseau, le routage ne nécessite qu'un seul message). Chord parvient à trouver un compromis entre la taille de la table de routage, comportant O(log(N)) entrées, et le routage nécessitant O(log(N)) messages. Pour cela, un nœud, ayant pour ID currentID, choisit chaque pointeur de sa table de manière à ce que le nœud sélectionné soit le seul représentant de l'intervalle [(currentID + $2i$) - (currentID + $2i+1$)] pour $i+1 < m$. Ainsi, chaque nœud connaît un successeur qui représente une portion de l'anneau deux fois plus grande à chaque nouvelle entrée.

Le routage est itératif et progressif, chaque nœud envoyant la requête à son successeur connu le plus proche de l'identifiant recherché.

Les principales limitations de Chord résident d'une part dans sa topologie en anneau, qui nécessite dans le pire des cas de parcourir l'ensemble de l'espace d'adressage, et d'autre part dans l'absence de primitives permettant de gérer une DHT. Ces lacunes sont résolues par l'architecture de Kademlia.

2.5.3 Modèle hybride

Une architecture hybride dans les réseaux peer-to-peer ($P2P$) combine les avantages des architectures centralisées et décentralisées pour améliorer les performances, la robustesse et l'efficacité des réseaux de partage de fichiers.

Les réseaux hybrides font appel à un nombre suffisant de serveurs (appelés super-pairs) pour minimiser les risques en cas de disparition de l'un d'eux. Chaque serveur constitue un nœud central d'un groupe (cluster) composé d'un ensemble de pairs organisés en $P2P$.

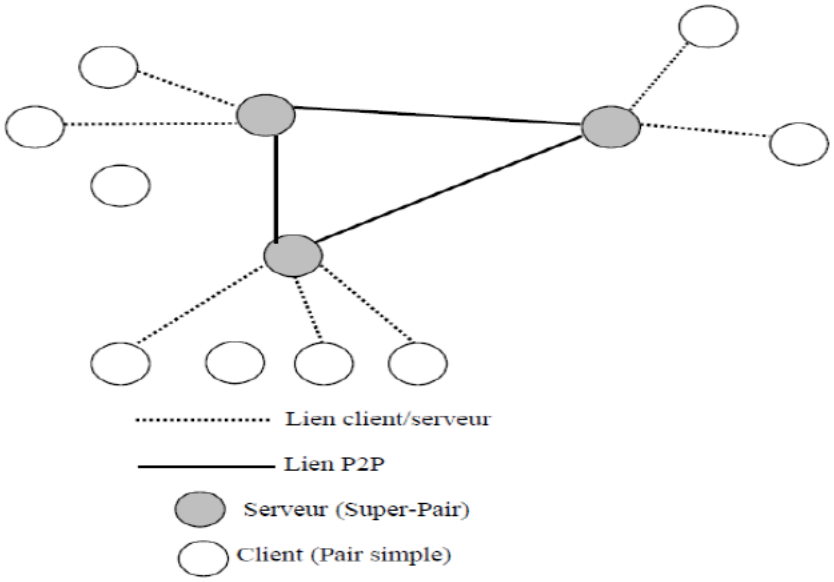

FIGURE 2.4 – Modèle hybride ou Super pair

Il existe deux types de réseaux hybrides :

Les hybrides statiques : Les super-pairs sont désignés manuellement dès le début. Ces nœuds peuvent à la fois agir en tant que clients, comme c'est le cas dans l'exemple du réseau eDonkey [6].

Les hybrides dynamiques : Dans certaines situations, le logiciel client peut décider de promouvoir un nœud client en super-pair, comme cela se produit dans l'exemple de Kazaa [6].

2.5.4 Comparaison des architectures $P2P$

Les trois tableaux présentent une étude comparative des architectures $P2P$ suivantes : $P2P$ centralisé, $P2P$ décentralisé non structuré et $P2P$ décentralisé stucturé [1].

Bienfaits	Méfaits
Recherche évaluées	Disponibilité du serveur
Charge minimale sur les pairs	Coût du serveur
Accès aux ressources	Confiance dans l'administration du serveur

TABLE 2.2 – Bienfaits et Méfaits du modèle centralisée

Bienfaits	Méfaits
Distribution complète	Passage à l'échelle limité (inondation)
Décentralisation	Charge non-homogène des pairs (super pairs)
Recherches non-exactes	

TABLE 2.3 – Bienfaits et Méfaits du modèle décentralisée non structurée

Bienfaits	Méfaits
Distribution complète Passage à l'échelle / Charge homogène Stockage des données dans le DHT Décentralisation	Recherche éxactes uniquement Structure à maintenir

Table 2.4 – Bienfaits et Méfaits du modèle décentralisée structurée

2.6 Bienfaits et Méfaits du Peer to Peer

Le peer-to-peer ($P2P$) présente plusieurs bienfaits et méfaits :

2.6.1 Bienfaits

En se basant sur les propriétés mentionnées ci-dessus, on peut affirmer que les systèmes P2P sont mieux adaptés aux environnements qui disposent de grandes quantités de ressources à partager.
•Évoluer vers une échelle supérieure.
•Prévenir les blocages du trafic.
•Utiliser efficacement des ressources significatives réparties sur de nombreux nœuds du réseau.
•Optimiser l'exploitation de la bande passante, des capacités de traitement et des espaces de stockage.
•Accélérer l'exécution des tâches en réduisant le temps de traitement grâce aux connexions directes entre les pairs.
•Prévenir les points de défaillance uniques en assurant une distribution redondante des ressources et en décentralisant les systèmes $P2P$. Cette caractéristique confère à ce type de systèmes une plus grande fiabilité et robustesse.
•Améliorer les performances du système en répartissant la charge de travail et en renforçant l'autonomie et l'agrégation des ressources, ce qui améliore les performances des réseaux $P2P$.
•Permet aux utilisateurs de conserver le contrôle de leurs ressources, tout en leur offrant la possibilité de rejoindre ou de quitter le système à tout moment.

2.6.2 Méfaits

Bien que les réseaux P2P soient largement utilisés aujourd'hui, ils présentent néanmoins des méfaits. Nous en mentionnons quelques-uns ici :

- **Les problèmes de comportement des utilisateurs :** Les systèmes d'échange de fichiers peer-to-peer peuvent être vulnérables à l'anarchie générale. En raison de l'anonymat qu'ils offrent, les utilisateurs sont parfois tentés d'adopter des comportements malveillants. En l'absence de contrôle adéquat, des activités telles que la diffusion de virus ou le freeloading peuvent se manifester.
- **Les problèmes de comportement des entreprises :**

  • Une atteinte a la vie privée
  Notre vie privée n'est pas toujours garantie lorsque nous utilisons des outils peer-to-peer. En effet, les adresses IP des utilisateurs peuvent être récupérées lorsque le logiciel est centralisé.

  • La pollution des réseaux Plusieurs entreprises proposent discrètement de polluer les réseaux d'échange de musique entre particuliers en y intégrant des fichiers de moindre qua-

lité ou incorrects, sans même annoncer d'ultimatum. Cette pratique vise à rendre moins attrayants ces réseaux gratuits et anarchiques. Quelques sociétés indépendantes développent en effet de nouvelles technologies dans le but de polluer ces réseaux. Leurs clients potentiels sont les industries de la musique et du cinéma, qui cherchent par tous les moyens à détourner les utilisateurs des téléchargements gratuits vers des systèmes sécurisés et payants.

- **La propagande :** Il est bien connu que la plupart des systèmes peer-to-peer sont envahis par des bannières publicitaires, même si désormais des versions allégées des logiciels sont proposées sans publicité.
- **Les réseaux pair à pair restent imparfaits à cause de :**
  •La confiance et le certificat.
  •L'anonymat.
  •La sécurité.
  •La performance.

2.7 Domaine d'appliaction des réseaux $P2P$

Les réseaux peer-to-peer ($P2P$) ont de nombreux domaines d'application, grâce à leur capacité à distribuer des tâches et des ressources sans nécessiter de serveur centralisé. Voici quelques domaines d'application majeurs des réseaux $P2P$:

2.7.1 Partage de fichiers

C'est l'une des applications les plus connues des réseaux $P2P$. Des plateformes comme BitTorrent [6] permettent aux utilisateurs de partager des fichiers volumineux de manière efficace et rapide.

2.7.2 Réseaux de diffusion de contenu

Les réseaux $P2P$ peuvent être utilisés pour distribuer du contenu multimédia, comme la musique, les vidéos et les jeux, réduisant ainsi la charge sur les serveurs centraux. Ces systèmes sont utilisés pour créer un réseau de nœuds permettant la recherche et le transfert de fichiers. Parmi ces systèmes, on trouve des exemples tels que Napster, Gnutella, KAZAA [6], Freenet [8] ...

2.7.3 Programmes de messagerie

Un programme de messagerie $P2P$ permet aux utilisateurs de communiquer directement entre eux sans passer par un serveur centralisé. Chaque utilisateur (ou nœud) dans le réseau agit à la fois comme client et serveur, ce qui peut offrir une meilleure résistance aux pannes et une plus grande confidentialité.

2.7.4 Streaming $P2P$

Le streaming $P2P$ (peer-to-peer) est une technologie de diffusion de contenu multimédia en temps réel qui s'appuie sur un réseau décentralisé. Contrairement au modèle client-serveur traditionnel, où un serveur centralisé diffuse le contenu à plusieurs clients, le streaming permet à chaque participant (ou pair) de partager des parties du contenu avec d'autres participants, réduisant ainsi la charge sur un seul serveur et améliorant l'efficacité et la résilience du réseau.

2.8 Conclusion

Dans ce chapitre, nous avons présenté divers types de réseaux $P2P$, à savoir les réseaux centralisés, hybrides, décentralisés, ainsi que les réseaux structurés et non structurés. nous avons présenté des illustrations de leurs structures spécifiques, ainsi que les bienfaits et les méfaits de chaque type. Nous avons également précisé leurs buts et leurs principales propriétés.

Dans le chapitre suivant, nous explorerons des modèles analytiques pour évaluer et optimiser les performances de ces réseaux.

Chapitre 3

Evaluation de Performance

3.1 Introduction

Dans le système informatique en particulier, où dans n'importe quel système en général, la mesure des performances de ce dernier est devenue essentielle. Cette démarche vise à concevoir un système qui répond aux différents objectifs mentionnés dans le cahier des charges. La mesure des performances consiste donc à identifier pour un système donné ses points forts et ses points faibles. Parmi les outils utilisés pour modéliser et évaluer les performances d'un réseau $P2P$ sont les modèles analytiques tels que les réseaux de Petri et les chaînes de Markov.

Nous abordons, dans ce chapitre, d'une part, les notions associées à la mesure des performances, et d'autre part, les modèles analytiques appliqués aux réseaux $P2P$.

3.2 Mesure de performance

La mesure des performances vise à analyser l'efficacité et la fiabilité d'un système $P2P$ pour identifier les améliorations possibles.

3.2.1 Importance de la Mesure des Performances

Toutes les applications informatiques ont le même objectif principal qui est de réaliser les fonctionnalités indiquées dans le cahier des charges, tout en maintenant une performance optimale avec des coûts maîtrisés. L'évaluation de la performance peut s'effectuer à deux niveaux.
Conception : Concevoir un système qui respecte les spécifications du cahier des charges.

Exploitation : Faisabilité des mises à jour du système déjà existant pour améliorer ses performances.

3.2.2 Phases de Mesure des Performances

Les phases de mesure de performance comprennent généralement la définition des objectifs de performance, la sélection des métriques pertinentes, la collecte des données, l'analyse des résultats et la prise de mesures correctives. Ces phases sont essentielles pour évaluer l'efficacité d'un système ou d'une application et identifier les domaines nécessitant des améliorations.

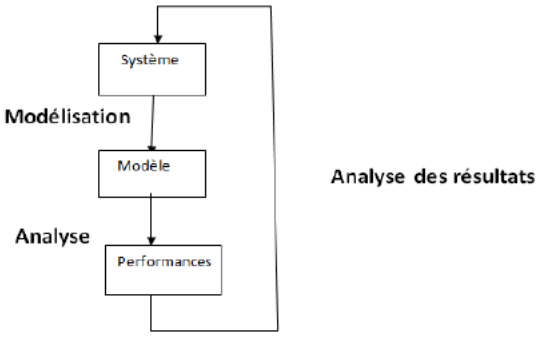

FIGURE 3.1 – Phases de mesure des performances pour un système

3.2.3 Approches d'évaluation

Les diverses approches d'évaluation des performances d'un système informatique se répartissent en deux catégories : les mesures directes sur le système et la modélisation.

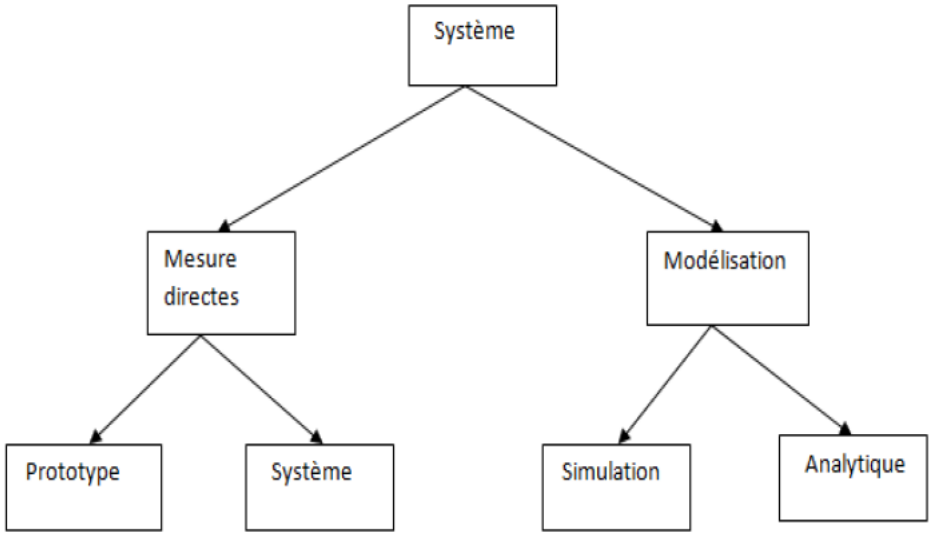

FIGURE 3.2 – Méthodes de mesure de performance

Les Mesures Directes

Cette méthode consiste à mesurer directement des paramètres de performance spécifiques du réseau, tels que le débit, le temps de réponse, le taux de perte de paquets, etc., afin d'évaluer son efficacité et sa qualité de service.

La modélisation

Cette méthode consiste à créer un modèle mathématique ou informatique du réseau, en utilisant des outils tels que les réseaux de Petri, les files d'attente, les simulations informatiques, etc. Une fois le modèle créé, on peut l'utiliser pour simuler le comportement du réseau dans différentes conditions et analyser sa performance. On distingue deux méthodes de Modélisation :

• La simulation

Cette technique repose sur l'utilisation d'un programme de simulation pour observer le comportement d'un modèle simplifié qui représente un système réel. Les résultats obtenus sont présentés sous forme de graphiques, permettant une analyse et une interprétation faciles. La méthode de simulation offre une représentation plus fidèle du système réel que les méthodes analytiques et est utilisée lorsque les évaluations directes sont coûteuses.

• Les Méthodes Analytiques

Les méthodes analytiques de modélisation sont des approches qui utilisent des outils mathématiques pour décrire et analyser le comportement des systèmes, y compris les réseaux de communication. Voici quelques-unes des méthodes analytiques couramment utilisées pour la modélisation des réseaux :

- Les réseaux de Petri sont employés pour la synchronisation des processus en temps réel ;
- Les réseaux de files d'attente sont utilisés pour modéliser les réseaux de communication ;
- Les algèbres des processus stochastiques sont employées lorsqu'il s'agit d'un système trop complexe ;
- Les modèles de Markov, qu'elles soient à temps discret ou continu, sont utilisées pour modéliser l'accès aux différentes ressources.

3.3 Méthodes analytiques

Divers modèles analytiques existent pour l'évaluation et l'analyse des performances des systèmes $P2P$, comme les réseaux de Petri et les chaînes de Markov.

En général, chaque modèle évalue un cas spécifique pour fournir un aperçu des performances des systèmes $P2P$.

3.3.1 Les Chaînes de Markov

Pour évaluer les performances des systèmes $P2P$, notamment ceux dynamiques discrets, les chaînes de Markov sont souvent employées. Ces modèles stochastiques permettent de déterminer les états possibles du système à un moment futur aléatoire et d'analyser les états stationnaires. Les probabilités associées à ces états stationnaires indiquent la proportion de temps que le système passe dans chaque état sur une longue période d'observation. Un système est considéré comme ayant un état stationnaire s'il converge vers ces probabilités après un certain temps, fournissant ainsi une vision stable de son comportement à long terme.

Deux catégories de chaînes de Markov existent.

Chaîne de Markov à temps continu (CMTC)

Un processus stochastique $\{X(t), t \geq 0\}$ une chaîne est dite de Markov à temps continu si elle vérifie les trois conditions suivantes :

1. Le processus $X(t)$ vérifie la propriété de Markov ;
2. Le temps d'observation est de nature continue ;
3. L'espace d'états S est dénombrable.

Une chaîne de Markov homogène à temps continu est définie par ses probabilités de transition :

$$P[X(t_n) = j/X(t_{n-1}) = i_{n-1}, ..., X(t_0) = i_0] = P[X(t_n) = j/X(t_{n-1}) = i_{n-1}], t_0 < t_1 < ... < t_n, n \in \mathbb{N} \tag{3.1}$$

$$p_{ij} = P[X(s+t) = j/X(s) = i],\ s \geq 0 \tag{3.2}$$

C-à-d que les probabilités $P[X(t_n) = j/X(t_{n-1}) = i]$ ne dépendent pas des instants d'obeservations t_{n-1},t_n, mais uniquement de la durée $t_n - t_{n-1}$ qui sépare les deux observations.

Chaîne de Markov à Temps Discret (CMTD)

Soit $\{(X_n), n = 0, 1, 2, ...\}$ un processus stochastique (P.S) à temps discret. Ce P.S est une chaîne de Markov à temps discret, si pour tout $n \in \mathbb{N}$ et quel que soient les $(n+1)$ états $i_0, i_1, ..., i_{n-1}, i, j$ de E tel que :

$$P[X(n+1) = j, X(n) = i, ..., X(0) = i_0] \tag{3.3}$$

La propriété de Markov est satisfaite :

$$P[X(n+1) = j/X(n) = i, ..., X(0) = i_n] = P[X(n+1) = j/X(n) = i] = p_{ij} \tag{3.4}$$

$p_{ij}(n)$ représente la probabilité de transition de l'état i vers l'état j à l'instant n.

L'état futur de la chaîne à l'instant $(n+1)$ dépend uniquement de l'état présent à l'instant n, et non de l'état par laquelle la chaîne de Markov est passée dans le passé (absence de mémoire).

Une chaîne de Markov est dite homogène si :

$$\forall n > 0,\ P[X(n+1) = j/X(n) = i] = p_{ij} \tag{3.5}$$

est indépendante de l'instant n.

La matrice de transition P est la matrice des $P = (p_{ij})\ /i, j \in$ E (si E est fini).

$$P = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0j} & \cdots \\ p_{10} & p_{11} & \cdots & p_{1j} & \cdots \\ \vdots & \cdots & \cdots & \cdots & \cdots \\ p_{i0} & p_{i1} & \cdots & p_{ij} & \cdots \\ \vdots & \vdots & \cdots & \vdots & \vdots \end{bmatrix}$$

Une chaîne de Markov à temps discret est entièrement définie par sa matrice de transition.

3.3.2 Notions de files d'attente

Les files d'attente sont un aspect typique de la vie moderne, elle sont présentes dans de nombreux domaines d'activité tels que les guichets de poste, le trafic routier, où les centres d'appels téléphoniques. L'analyse mathématique des files d'attente constitue un aspect fondamental des processus stochastiques. Les files d'attente décrivent des situations où des "clients" (ou unités) arrivent de manière aléatoire à des "serveurs" pour obtenir un service de durée variable.

Processus d'arrivée

L'arrivée des clients à la station est modélisée par un processus stochastique N_t de comptage $N(t), t \geq 0$. Dans le cas où la variable aléatoire A_n qui représente l'instnat d'arrivée de l'unité n (le n^{ieme} client) du système, on obtient :

- $A_0 = 0$
- $A_n =$inf
- $N_t =$n
- $T_n = A_n - A_{n-1}$, VA le temps entre l'arrivée du $(n-1)^{ieme}$ client et celle du n^{ieme} client

Temps de service

La distribution exponentielle est simple à étudier pour le temps de service, mais sa propriété "sans mémoire" la rend souvent peu réaliste pour modéliser les phénomènes réels. Cela conduit souvent à explorer d'autres distributions pour mieux représenter le temps de service.

Structure de file

Nombre de serveurs : Plusieurs serveurs peuvent être mis en parallèle dans une station (on représente le nombre de ces serveurs par C). Lorsque le client c_i arrive à une station, soit un serveur est disponible et le client est immédiatement pris en charge, soit tous les serveurs sont occupés et le client rejoint la file d'attente en attendant qu'un serveur se libère. On suppose que les serveurs, en général, sont identiques ayants la même configuration et distribution et sont indépendants entre eux.

Capacité de la file : a file d'attente peut avoir une capacité finie ou infinie. On note K la capacité de la file, qui inclut les clients en attente de service ainsi que ceux en cours de service. Une file avec une capacité illimitée est représentée par K=$+\infty$. Lorsque la capacité de la file est limitée et qu'un client arrive alors qu'elle est pleine, ce client est perdu.

Discipline de service : La discipline de service détermine l'ordre dans lequel les clients sont placés dans la file d'attente et en sont retirés pour recevoir un service. Les disciplines les plus fréquentes sont :

- **FIFO** (first in, first out) : premier arrivée, premier servi.
- **LIFO** (last in, first out) : dernier arrivé, premier servi.
- **FIRO** (first in, Random out) : les clients sont servis de manière aléatoire.

Notation de Kendall

La notation de Kendall permet de décrire un système de files d'attente à l'aide d'une série de paramètres. Pour identifier un système de files d'attente, le formalisme suivant a été proposé et est largement accepté : A/B/n/K/N

A : La première lettre identifie la loi du processus d'arrivée, tandis que la seconde lettre B identifie la loi du processus de service, avec les conventions suivantes dans les deux cas :

M :loi "sans mémoire" (arrivées poissoniennes, service exponentiel) ;

D :loi déterministe ;

Ek :loi "Erlang-k"

Hk : loi "hyper exponentielle" d'ordre k ;

GI : loi générale, les variables successives étant indépendantes ;

G : loi générale, sans hypothèse d'indépendance
n : Le chiffre qui suit donne le nombre de serveurs,
K et N : Les lettres suivantes (qui sont facultatives) identifient la taille de la file d'attente et la taille de la population source (si ces valeurs sont omises, elles sont supposées infinies).

Quelques mesures de performance :

L'analyse théorique d'un modèle de files d'attente vise à comprendre de manière quantitative et qualitative le fonctionnement du système en question. Pour ce faire, il est nécessaire de définir des critères et des mesures permettant d'atteindre cet objectif, afin de pouvoir déterminer à l'avance les performances du système, les effets d'un changement, ou encore d'identifier les paramètres les plus sensibles. Pour un système composé d'une seule file d'attente, les principales mesures de performance sont :

1. L : Nombre moyen de clients dans le système.
2. T : Temps de séjour d'un client dans le système.
3. L_q : Nombre moyen de clients dans la file d'attente.
4. W : Temps de séjour moyen d'un client dans le système.
5. W_q : Temps de séjour moyen d'un client dans la file d'attente.
6. Le taux d'occupation $\rho = \lambda/\mu$ d'un serveur.

Ses facteurs sont liés de la manière suivante :

- $L = \lambda W$
- $L_q = \lambda W_q$
- $W = W_q + 1/\mu$
- $L = \lambda/\mu + L_q$

Il est essentiel de noter que ces grandeurs sont aléatoires : leur espérance et leur distribution dépendent de l'instant t auquel elles sont considérées. Ainsi, nous devrions parler du nombre de clients dans le système à l'instant t.

Seulement en supposant la stationnarité du système, ces mesures de performance deviennent indépendantes du temps. Lors de l'étude d'une file d'attente, on cherche à estimer l'espérance en régime stationnaire des grandeurs précédentes.

Caractéristique du système

Les modèles markoviens de files d'attente sont des systèmes où les deux principales quantités stochastiques, à savoir le temps entre les arrivées et la durée du service, sont des variables aléatoires indépendantes et suivent une distribution exponentielle. La propriété "sans mémoire" de la loi exponentielle facilite l'analyse de ces modèles.

Système M/M/1

Pour le système d'attente M/M/1, le flux des arrivées est poissonien de paramètre λ et la durée de service est exponentielle de paramètre μ. La capacité d'attente est illimitée et il y a une seule station de service.
Si $\rho = \lambda/\mu < 1$, on dit que le système est stable.

Système M/M/$+\infty$

On considère un système composé d'un nombre illimité de serveurs identiques et indépendants les uns des autres. Dès qu'un client arrive, il rentre donc instantanément en service. Danc cette file particulière, il n'y a donc pas d'attente. On suppose toujours que le processus d'arrivée des clients est poissonien de taux λ et que les temps de service sont exponentiels de taux μ (pour tous les serveurs).

Système M/M/1/k

Le modèle M/M/1/K correspond au cas où la capacité est de K clients, où K représente le nombre maximal de clients dans le système, c'est-à-dire en attente et en service.
Si un client arrive et trouve K clients déjà dans le système, ce client sera perdu. Ainsi, la probabilité de perte est égale à la probabilité de trouver K clients dans le système en régime stationnaire, on a : $p_{perte} = \rho^k(1-\rho)/1-\rho^{k-1}$

Modèle d'attente non-Markovien

En abandonnant l'hypothèse d'expontentialité pour l'une des deux quantités stochastiques, que ce soit le temps entre les arrivées ou les durées des services, ou en introduisant des paramètres supplémentaires spécifiques, le processus ne sera plus markovien. Cela rend l'analyse du modèle analytique très délicate, voire impossible. C'est pourquoi, souvent, on simplifie en se ramenant à un processus markovien.

Système G/G/1

Pour décrire l'évaluation du système G/G/1, il est nécessaire de caractériser, en plus du nombre de clients dans le système, le temps déjà passé dans le serveur par le client en service et le temps écoulé depuis l'arrivée du dernier client.

Système M/G/1

Une file d'attente M/G/1 est un modèle de file d'attente avec des arrivées aléatoires selon un processus de Poisson, des temps de service avec une distribution quelconque, et un seul serveur pour traiter les clients.

Théorie des réseaux de files d'attente

Un réseau de files d'attente est un ensemble de files d'attente interconnecté que l'on classe en deux catégorie.
Réseau monoclassé : Tous les clients ont le même comportement aléatoire. Tous les cients sont statistiquement indistinguables.
Réseau multiclassé : Les cients de différente classe ont de différentes caractéristiques de temps de service et ont des parcours à trouver le réseau. Tous les clients d'une même classe sont statistiquement indistinguable.

Les défférentes types de réseaux

Deux types de réseaux sont distingués :

Réseaux de Jackson ouverts : Ces réseaux ont une ou plusieurs entrées externes et peuvent accueillir un nombre illimité de clients à tout moment.
Réseaux de Jackson fermés : Ces réseaux n'ont pas de connexions externes, donc le nombre de clients est constant. Dans le cas d'un routage probabiliste, la probabilité que qu'un client quitte la station i pour se rendre à la station j est définie par p_{ij}.

3.4 Evaluation des indices de performances

Une fois le modèle établi, on examine ses propriétés qualitatives pour déduire son ergodicité et effectuer une analyse quantitative. Si le modèle est ergodique, alors la distribution de probabilité des

marquages des états stationnaires existe et est unique.
Plusieurs mesures de performance peuvent être calculées. Parmi les plus importants, on trouve :

• **Fréquence moyenne de franchissement d'une transition :** La fréquence moyenne (débit moyen) de franchissement d'une transition t_i est le nombre moyen de tirs par t_i en une unité de temps. Elle est calculée en

$$\bar{\lambda}(t_j) = \sum_{M_j \in E(t_j)} \lambda_i(M_j)\pi_j \tag{3.6}$$

où :
$E(t_i)$ est l'ensemble des marquages où la transition t_i est franchissable .
$\lambda(M_j)$ est le taux de franchissement de t_i en M_j.

• **Nombre moyen de marques dans une place :** Le nombre moyen de marques dans une place p est calculé en appliquant la formule :

$$n(p) = \sum_{i:M_j \in E} M_i(p)\pi_i \tag{3.7}$$

où :
$M_i(p)$ est le nombre moyen de jetons dans la place p pour le marquage M_i.
E est l'ensemble des marquages accessibles.

• **Le temps moyen de séjour d'une marque dans un sous-réseau :** Le temps moyen qu'un jeton passe dans une partie S (sous-réseau) d'un RdPSG à l'état stationnaire peut être calculé en utilisant la formule de Little :

$$E[T] = \frac{E[N]}{E[\beta]} \tag{3.8}$$

où :
$E[T]$: est le nombre moyen de jetons dans S.
$E[\beta]$: est le taux d'arrivée effectif des jetons dans S.

3.5 Conclusion

Dans ce chapitre, nous avons exploré divers modèles analytiques performants pour la modélisation des réseaux $P2P$, en décrivant leurs paramètres et caractéristiques. Chaque outil est adapté à un contexte spécifique, et le choix du modèle repose sur plusieurs critères, tels que les performances à évaluer.

Chapitre 4

Modélisation et application

4.1 Introduction

Ce chapitre présente un cadre de simulation de réseaux pair-à-pair (P2P) en utilisant Python, avec les bibliothèques networkx, simpy, random et matplotlib. Le but est de modéliser et de simuler différentes topologies de réseaux et d'étudier le comportement des requêtes de recherche, les échecs de nœuds, la réplication de fichiers, et la congestion dans ces réseaux. Nous utilisons également des matrices de transition et des chaînes de Markov pour analyser les dynamiques du réseau.

4.2 Python

Python est un language de programmation largement utilisé aujourd'hui dans divers domaine d'application, comme le développement web, l'analyse de données, l'intelligence artificielle, l'apprentissage automatique et bien plus encore. Sa simplicité syntaxique et sa lisibilité en font un choix populaire pour les débutants.

Python est connue pour sa bibliothèque standard étendue, qui fournit des modules et des packages pour des tâches variées comme le traitement de fichiers, le réseau, les bases de données, les mathématiques,etc.

Python joue un rôle important dans le développement et l'analyse des réseaux peer-to-peer (P2P). Grâce à ses bibliothèques tels que : simpy, numpy, et frameworks, Python permet de créer, simuler et optimiser des réseaux P2P.

4.3 La modélisation d'un réseau $P2P$ structurée

le code que nous avons utilisé simule différentes topologies de réseaux et étudie divers aspects de leur performance dans le cadre d'un réseaus $P2P$, on a utilisé les bibliothèques (networkx, simpy, numpy). Utilisons les bibliothéques définies dans une application :

4.3.1 Création de la topologie de réseaux

Cette fonction crée un réseau en anneau où chaque nœud est connecté à ses voisins immédiats, formant un cycle.
• **'create_ring_topology(n)' :** Crée une topologie en anneau avec 'n' nœuds .

• **‘create_tree_topology(n)‘ :** Crée une topologie en arbre équilibré avec ’n’ nœuds.
• **‘create_grid_topology(m,n)‘ :** Crée une topologie en grille avec ’m’ lignes et ’n’ colonnes.

```python
# Création d'un réseau en anneau
def create_ring_topology(n):
    G = nx.Graph()
    for i in range(n):
        G.add_node(i)
        G.add_edge(i, (i+1) % n)
    return G

# Création d'un réseau en arbre
def create_tree_topology(n):
    G = nx.balanced_tree(2, int(n/2))
    return G

# Création d'un réseau en grille
def create_grid_topology(m, n):
    G = nx.grid_2d_graph(m, n)
    return G
```

FIGURE 4.1 – Création des topologies en anneau, en arbre, en grille

4.3.2 Simulation de requêtes de recherche

• **’search_request(env, G, start_node, target_file, search_time)’ :** simule une requête de recherche dans le réseau depuis ’start_node’ vers ’target_file’, prenant ’search_time’ unités de temps.
• **‘run_simulation(topology, num_nodes, search_time)‘ :** Initialise et exécute une simulation de requêtes de recherche pour une topologie donnée (’ring’, ’tree’, ’grid’).

```python
# Fonction pour envoyer une requête de recherche
def search_request(env, G, start_node, target_file, search_time):
    yield env.timeout(search_time)
    # Simuler la recherche dans le réseau
    path = nx.shortest_path(G, source=start_node, target=target_file)
    print(f"Requête de {start_node} à {target_file} via {path}")

# Initialisation de la simulation
def run_simulation(topology, num_nodes, search_time):
    if topology == "ring":
        G = create_ring_topology(num_nodes)
    elif topology == "tree":
        G = create_tree_topology(num_nodes)
    elif topology == "grid":
        G = create_grid_topology(int(num_nodes**0.5), int(num_nodes**0.5))

    env = simpy.Environment()
    for i in range(num_nodes):
        target_file = random.choice(list(G.nodes))
        env.process(search_request(env, G, i, target_file, search_time))

    env.run()

# Exemple d'exécution de la simulation
run_simulation("ring", 10, 2)
```

FIGURE 4.2 – La fonction pour envoyer une requête

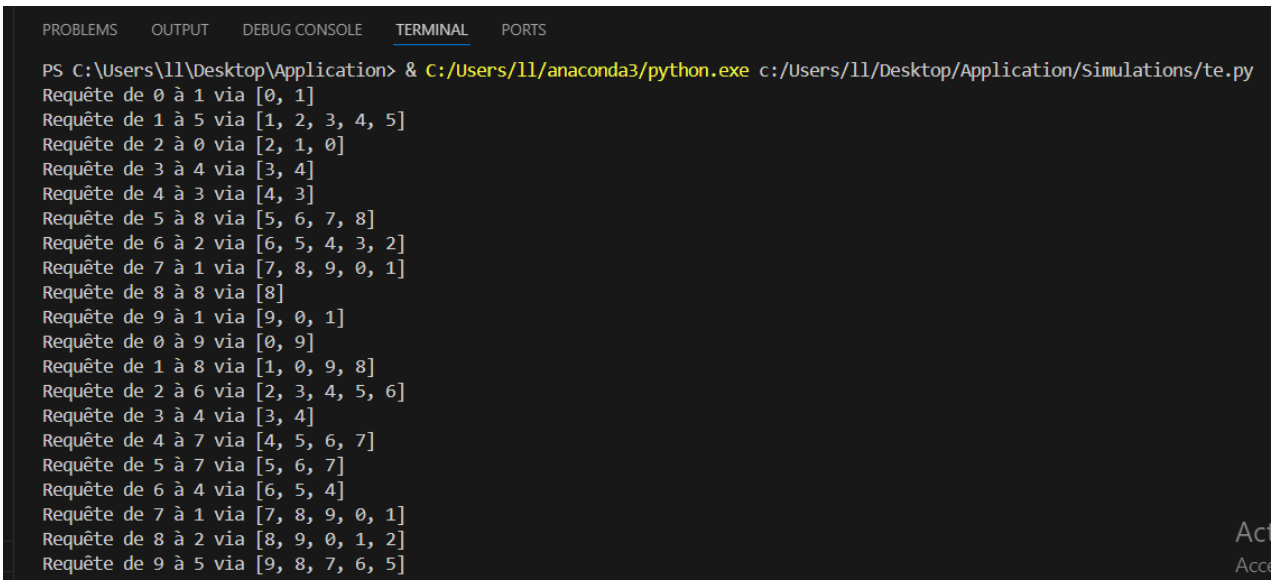

FIGURE 4.3 – Les résultats de la simulation

4.3.3 Simulation de pannes de nœuds

• **'node_failure(env, G, node, failure_time)' :** Simule la panne d'un nœud spécifique après 'failure_time' unités de temps.
• **'run_failure_simulation(topology, num_nodes, search_time, failure_time)' :** Exécute une simulation avec pannes de nœuds en plus des requêtes de recherche.
Lorsque des nœuds échouent, la topologie du réseau change. Les connexions associées aux nœuds défaillants sont supprimées, ce qui peut entraîner une fragmentation du réseau, rendant certains nœuds ou ressources inaccessibles (voir la figure 4.5).

```python
def node_failure(env, G, node, failure_time):
    yield env.timeout(failure_time)
    G.remove_node(node)
    print(f"Node {node} a échoué à l'heure {env.now}")

def run_failure_simulation(topology, num_nodes, search_time, failure_time):
    if topology == "ring":
        G = create_ring_topology(num_nodes)
    elif topology == "tree":
        G = create_tree_topology(num_nodes)
    elif topology == "grid":
        G = create_grid_topology(int(num_nodes**0.5), int(num_nodes**0.5))

    env = simpy.Environment()
    for i in range(num_nodes):
        target_file = random.choice(list(G.nodes))
        env.process(search_request(env, G, i, target_file, search_time))

    for i in range(num_nodes // 2):  # Simuler des pannes pour la moitié des nœuds
        env.process(node_failure(env, G, i, failure_time))

    env.run()

# Exemple d'exécution de la simulation
run_failure_simulation("ring", 10, 2, 5)
```

FIGURE 4.4 – Les fonctions pour simuler les pannes de nœuds

```
Node 0 a échoué à l'heure 5
Node 1 a échoué à l'heure 5
Node 2 a échoué à l'heure 5
Node 3 a échoué à l'heure 5
Node 4 a échoué à l'heure 5
```

FIGURE 4.5 – Les résultats de la simulation

4.3.4 Simulation de réplication de fichiers

• **'replicate_file(env, G, file_node, replication_strategy)' :** Simule la réplication d'un fichier à partir d'un nœud donné selon une stratégie de réplication ('random','closest')
• **'run_replication_simulation(topology, num_nodes, search_time, replication_strategy)' :** Exécute une simulation de réplication de fichiers en plus des requêtes de recherche.
La partie de réplication du script vise à améliorer la disponibilité des fichiers dans le réseau en répliquant les fichiers sur plusieurs nœuds. Cela permet de garantir que les fichiers restent accessibles même en cas de défaillance de certains nœuds. Les stratégies de réplication peuvent être ajustées pour répliquer les fichiers de manière aléatoire ou sur les nœuds les plus proches, en fonction des besoins spécifiques du réseau.

```python
def replicate_file(env, G, file_node, replication_strategy):
    yield env.timeout(1)
    if replication_strategy == "random":
        replicas = random.sample(list(G.nodes), k=3)
    elif replication_strategy == "closest":
        replicas = sorted(G.nodes, key=lambda x: nx.shortest_path_length(G, source=file_node, target=x))[:3]

    for replica in replicas:
        G.nodes[replica]['file'] = True
        print(f"Fichier {file_node} répliqué sur {replica}")

def run_replication_simulation(topology, num_nodes, search_time, replication_strategy):
    if topology == "ring":
        G = create_ring_topology(num_nodes)
    elif topology == "tree":
        G = create_tree_topology(num_nodes)
    elif topology == "grid":
        G = create_grid_topology(int(num_nodes**0.5), int(num_nodes**0.5))

    for node in G.nodes:
        G.nodes[node]['file'] = False

    env = simpy.Environment()
    file_node = random.choice(list(G.nodes))
    env.process(replicate_file(env, G, file_node, replication_strategy))

    for i in range(num_nodes):
        target_file = random.choice(list(G.nodes))
        env.process(search_request(env, G, i, target_file, search_time))

    env.run()

# Exemple d'exécution de la simulation
run_replication_simulation("ring", 10, 2, "random")
```


FIGURE 4.6 – Simulation de réplication de fichiers

```
Fichier 2 répliqué sur 4
Fichier 2 répliqué sur 8
Fichier 2 répliqué sur 0
```

FIGURE 4.7 – Les résultats de la simulation

4.3.5 Simulation de congestion

• **‘search_request_with_congestion(env, G, start_node, target_file, search_time)‘ :** Simule une requête de recherche en prenant en compte la congestion (augmentation du trafic) sur les nœuds.
•**’run_congestion_simulation(topology, num_nodes, search_time)’ :** Exécute une simulation de congestion pour une topologie donnée.

```python
def search_request_with_congestion(env, G, start_node, target_file, search_time):
    yield env.timeout(search_time)
    path = nx.shortest_path(G, source=start_node, target=target_file)
    for node in path:
        if 'congestion' not in G.nodes[node]:
            G.nodes[node]['congestion'] = 0
        G.nodes[node]['congestion'] += 1

    print(f"Requête de {start_node} à {target_file} via {path} avec congestion {G.nodes[node]['congestion']}")

def run_congestion_simulation(topology, num_nodes, search_time):
    if topology == "ring":
        G = create_ring_topology(num_nodes)
    elif topology == "tree":
        G = create_tree_topology(num_nodes)
    elif topology == "grid":
        G = create_grid_topology(int(num_nodes**0.5), int(num_nodes**0.5))

    env = simpy.Environment()
    for i in range(num_nodes * 2):  # Plus de requêtes pour simuler la congestion
        target_file = random.choice(list(G.nodes))
        env.process(search_request_with_congestion(env, G, i % num_nodes, target_file, search_time))

    env.run()

# Exemple d'exécution de la simulation
run_congestion_simulation("ring", 10, 2)
```

FIGURE 4.8 – Les fonctions pour la simulation de congestion

```
Requête de 0 à 7 via [0, 9, 8, 7]
Requête de 1 à 8 via [1, 0, 9, 8]
Requête de 2 à 8 via [2, 1, 0, 9, 8]
Requête de 3 à 3 via [3]
Requête de 4 à 4 via [4]
Requête de 5 à 4 via [5, 4]
Requête de 6 à 7 via [6, 7]
Requête de 7 à 0 via [7, 8, 9, 0]
Requête de 8 à 7 via [8, 7]
Requête de 9 à 5 via [9, 8, 7, 6, 5]
Requête de 0 à 0 via [0] avec congestion 1
Requête de 1 à 4 via [1, 2, 3, 4] avec congestion 1
Requête de 2 à 9 via [2, 1, 0, 9] avec congestion 1
Requête de 3 à 8 via [3, 2, 1, 0, 9, 8] avec congestion 1
Requête de 4 à 7 via [4, 5, 6, 7] avec congestion 1
Requête de 5 à 9 via [5, 6, 7, 8, 9] avec congestion 3
Requête de 6 à 2 via [6, 5, 4, 3, 2] avec congestion 4
Requête de 7 à 5 via [7, 6, 5] avec congestion 4
Requête de 8 à 0 via [8, 9, 0] avec congestion 4
Requête de 9 à 2 via [9, 0, 1, 2] avec congestion 5
Requête de 0 à 0 via [0] avec congestion 6
Requête de 1 à 3 via [1, 2, 3] avec congestion 4
Requête de 2 à 0 via [2, 1, 0] avec congestion 7
Requête de 3 à 2 via [3, 2] avec congestion 8
Requête de 4 à 4 via [4] avec congestion 4
Requête de 5 à 5 via [5] avec congestion 5
Requête de 6 à 0 via [6, 7, 8, 9, 0] avec congestion 8
Requête de 7 à 5 via [7, 6, 5] avec congestion 6
Requête de 8 à 8 via [8] avec congestion 5
Requête de 9 à 8 via [9, 8] avec congestion 6
```

FIGURE 4.9 – Les résultats de la simulation

4.3.6 Analyse de la chaîne de Markov

• **‘create_transition_matrix_ring(num_nodes)‘ :** Crée une matrice de transition pour une topologie en anneau.
• **‘stationary_distribution(P)‘ :** Calcule la distribution stationnaire d’une matrice de transition donnée.
• **‘draw_markov_chain(P)‘ :** Visualise le graphe de la chaîne de Markov correspondant à la matrice de transition.

```python
import numpy as np

# Fonction pour créer une matrice de transition pour un réseau en anneau
def create_transition_matrix_ring(num_nodes):
    P = np.zeros((num_nodes, num_nodes))

    for i in range(num_nodes):
        P[i, (i+1) % num_nodes] = 0.5  # Transition vers le nœud suivant
        P[i, (i-1) % num_nodes] = 0.5  # Transition vers le nœud précédent

    return P

# Exemple d'utilisation
num_nodes = 10
P = create_transition_matrix_ring(num_nodes)
print(P)
```

FIGURE 4.10 – Fonction pour créer une matrice de transition

```
[[0.  0.5 0.  0.  0.  0.  0.  0.  0.  0.5]
 [0.5 0.  0.5 0.  0.  0.  0.  0.  0.  0. ]
 [0.  0.5 0.  0.5 0.  0.  0.  0.  0.  0. ]
 [0.  0.  0.5 0.  0.5 0.  0.  0.  0.  0. ]
 [0.  0.  0.  0.5 0.  0.5 0.  0.  0.  0. ]
 [0.  0.  0.  0.  0.5 0.  0.5 0.  0.  0. ]
 [0.  0.  0.  0.  0.  0.5 0.  0.5 0.  0. ]
 [0.  0.  0.  0.  0.  0.  0.5 0.  0.5 0. ]
 [0.  0.  0.  0.  0.  0.  0.  0.5 0.  0.5]
 [0.5 0.  0.  0.  0.  0.  0.  0.  0.5 0. ]]
```

FIGURE 4.11 – La matrice de transition

```python
import scipy.linalg

# Fonction pour calculer la distribution stationnaire
def stationary_distribution(P):
    evals, evecs = scipy.linalg.eig(P, left=True, right=False)
    evec1 = evecs[:, np.isclose(evals, 1)]

    # Normaliser la distribution stationnaire
    stationary = evec1 / evec1.sum()
    stationary = stationary.real.flatten()

    return stationary

# Exemple d'utilisation
stationary = stationary_distribution(P)
print("Distribution stationnaire:", stationary)
```

FIGURE 4.12 – La fonction pour calculer la distribution stationnaire

```
Distribution stationnaire: [0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]
```

FIGURE 4.13 – La distribution stationnaire

4.3.7 Simulation de génération de requêtes et analyse de performance

• **‘generate_requests(env, G, num_nodes, interarrival_time, search_time)‘ :** Génère des requêtes de recherche dans le réseau à des intervalles de temps moyens (‘interarrival_time‘).
• **‘search_request(env, G, start_node, target_file, search_time, request_times)‘ :** Simule une requête de recherche et enregistre le temps de réponse.
• **‘run_simulation(topology, num_nodes, interarrival_time, search_time, simulation_time)‘ :** Exécute la simulation principale pour une topologie donnée, en mesurant des métriques de performance comme le taux moyen effectif des arrivées, le temps moyen de réponse, et le nombre moyen

de requêtes dans le système.

```python
# Processus de génération de requêtes
def generate_requests(env, G, num_nodes, interarrival_time, search_time):
    request_times = []
    while True:
        start_node = random.choice(list(G.nodes))
        target_file = random.choice(list(G.nodes))
        env.process(search_request(env, G, start_node, target_file, search_time, request_times))
        yield env.timeout(interarrival_time)
```

FIGURE 4.14 – Processus de génération de requêtes

```python
# Processus de traitement des requêtes
def search_request(env, G, start_node, target_file, search_time, request_times):
    arrival_time = env.now
    yield env.timeout(search_time)
    path = nx.shortest_path(G, source=start_node, target=target_file)
    response_time = env.now - arrival_time
    request_times.append(response_time)
    print(f"Requête de {start_node} à {target_file} via {path} avec temps de réponse {response_time}")
```

FIGURE 4.15 – Processus de traitement des requêtes

```python
# Simulation principale
def run_simulation(topology, num_nodes, interarrival_time, search_time, simulation_time):
    if topology == "ring":
        G = create_ring_topology(num_nodes)

    env = simpy.Environment()
    request_times = []
    env.process(generate_requests(env, G, num_nodes, interarrival_time, search_time))
    env.run(until=simulation_time)
```

FIGURE 4.16 – Simulation principale

```
Requête de 5 à 4 via [5, 4] avec temps de réponse 2
Requête de 6 à 4 via [6, 5, 4] avec temps de réponse 2
Requête de 2 à 3 via [2, 3] avec temps de réponse 2
Requête de 3 à 6 via [3, 4, 5, 6] avec temps de réponse 2
Requête de 1 à 7 via [1, 0, 9, 8, 7] avec temps de réponse 2
Requête de 8 à 6 via [8, 7, 6] avec temps de réponse 2
Requête de 4 à 1 via [4, 3, 2, 1] avec temps de réponse 2
Requête de 7 à 9 via [7, 8, 9] avec temps de réponse 2
Requête de 8 à 6 via [8, 7, 6] avec temps de réponse 2
Requête de 9 à 0 via [9, 0] avec temps de réponse 2
Requête de 6 à 8 via [6, 7, 8] avec temps de réponse 2
Requête de 9 à 0 via [9, 0] avec temps de réponse 2
Requête de 5 à 4 via [5, 4] avec temps de réponse 2
Requête de 4 à 1 via [4, 3, 2, 1] avec temps de réponse 2
Requête de 7 à 7 via [7] avec temps de réponse 2
Requête de 6 à 7 via [6, 7] avec temps de réponse 2
Requête de 3 à 1 via [3, 2, 1] avec temps de réponse 2
Requête de 8 à 4 via [8, 7, 6, 5, 4] avec temps de réponse 2
Requête de 7 à 5 via [7, 6, 5] avec temps de réponse 2
Requête de 1 à 3 via [1, 2, 3] avec temps de réponse 2
Requête de 2 à 5 via [2, 3, 4, 5] avec temps de réponse 2
Requête de 0 à 4 via [0, 1, 2, 3, 4] avec temps de réponse 2
Requête de 3 à 7 via [3, 4, 5, 6, 7] avec temps de réponse 2
Requête de 6 à 8 via [6, 7, 8] avec temps de réponse 2
Requête de 7 à 6 via [7, 6] avec temps de réponse 2
Requête de 9 à 8 via [9, 8] avec temps de réponse 2
Requête de 0 à 6 via [0, 9, 8, 7, 6] avec temps de réponse 2
Requête de 4 à 9 via [4, 3, 2, 1, 0, 9] avec temps de réponse 2
```

FIGURE 4.17 – Résultats de génération de requêtes avec le temps de réponse

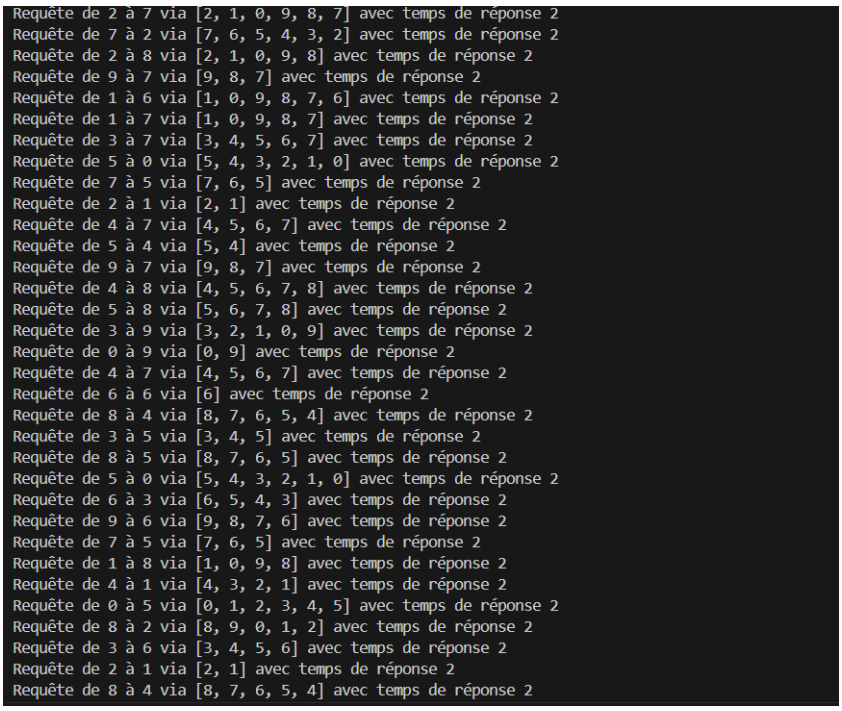

FIGURE 4.18 – Résultats de génération de requêtes avec le temps de réponse

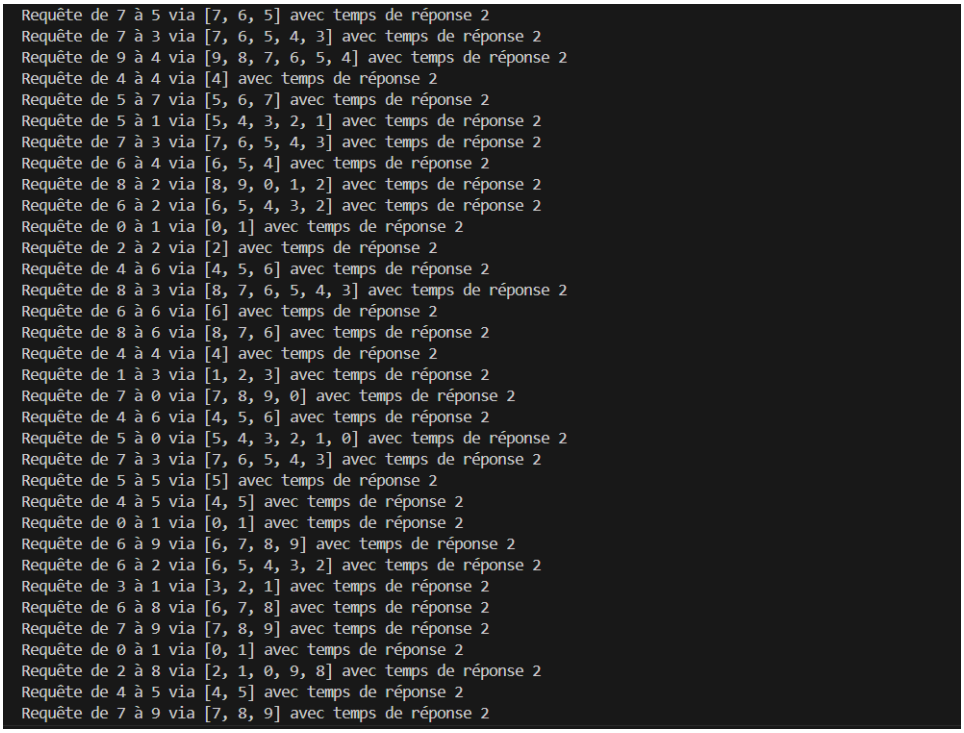

FIGURE 4.19 – Résultats de génération de requêtes avec le temps de réponse

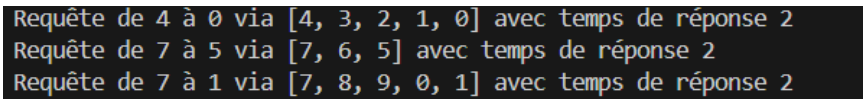

FIGURE 4.20 – Résultats de génération de requêtes avec le temps de réponse

4.3.8 Calcul des trois métriques de performance dans la simulation d'un réseau

• **'lambda_eff = len(request_times) / simulation_time' :** Calculer le taux moyen effectif des arrivées.

• **‘W = np.mean(request_times) if request_times else 0‘ :** Calculer le temps moyen de réponse.
• **‘L = lambda_eff * W‘ :** Calculer le nombre moyen de requêtes dans le système.

```python
# Calculer le taux moyen effectif des arrivées
lambda_eff = len(request_times) / simulation_time

# Calculer le temps moyen de réponse
W = np.mean(request_times) if request_times else 0

# Calculer le nombre moyen de requêtes dans le système
L = lambda_eff * W

return lambda_eff, W, L
```

FIGURE 4.21 – Calcul des trois métriques de performance

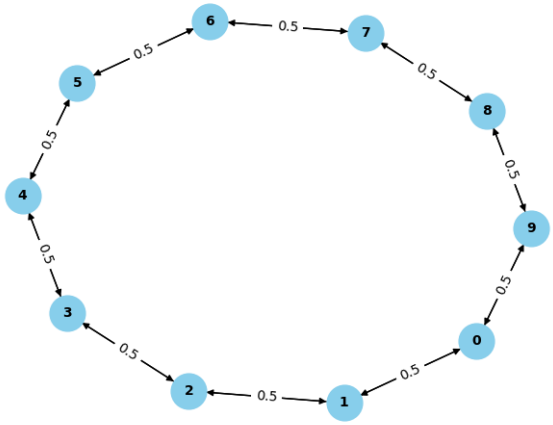

FIGURE 4.22 – Chaîne de Markov

Montrons les résultats d’une simulation :

```
Taux moyen effectif des arrivées (λ): 0.45
Temps moyen de réponse (W): 2.0
Nombre moyen de requêtes dans le système (L): 0.9
```

FIGURE 4.23 – le résultat des performances

Les métriques de perfomance sont :
Taux moyen effectif des arrivées : λ=0.45.
Temps moyen de réponse : W=2.0.
Nombre moyen de requêtes dans le système : L=0.45

4.4 Conclusion

Toutefois, pour obtenir une compréhension, plus complète et précise des performances des réseaux $P2P$, il est souvent nécessaire de recourir à des simulations numériques.

Les simulations permettent de tester la robustesse du réseau face à différentes conditions, ce qui est crucial pour concevoir et optimiser des réseaux $P2P$ capable de maintenir des performances optimales dans des environnements dynamiques et exigeants.

Conclusion Générale

Les réseaux $P2P$ représentent une stratégie innovante et performante pour le partage de ressources et la distribustion de charges dans un environnement décentralisé. Leur capacité à évoluer et à résister aux défaillances les rend particulièrement adapptés aux applications modernes exigeantes.

L'analyse des performmances des réseaux pair à pair $P2P$ structurées est cruciale pour garantir leur efficacité, leur robustesses et leur scalabilités, des aspects essentiels pour répondre aux besoins des applications modernes.

En intégrant les réseaux de Petri pour modéliser et analyser les comportements complexes des systèmes distribués, on peut capturer les interactions et flux de données de manière précise.
Les simulations numériques viennent compléter cette approche en offrant des résultats pratiques et concrets, illustrant des concepts théoriques par des exemples tels que le protocole chord.
Cette méthodologie combinée montre que les réseaux $P2P$ structurés peuvent assurer une recherche rapide avec une latence réduite, un équilibrage de charge efficace et une haute tolérance aux pannes, même en présence de défaillances de nœuds.

Ainsi, l'utilisation des réseaux de Petri et des simulations numériques se révèle indispensable pour concevoir et optimiser des réseaux $P2P$ structurés capables de maintenir des performances optimales dans des environnements dynamiques et exigeants.

Dans le cadre de ce travail, nous avons défini les différents types selon des critères de classification spécifques. En particulier, nous avons présenté les différents modèles analytiques proposés pour l'analyse des performances des systèemes $P2P$. Nous nous somme intéressés à l'analyse de performance du réseau pair à pair, notre but de faire une simulation en utilisant les réseaux de Petri.

Bibliographie

[1] D. Arkoub and Y. Krouri. "modèles analytiques pour l'évaluation des performances d'un réseau pair a pair (bittorrent)". *Université A/Mira de Béjaia*, 2013.

[2] G. Ciardo. Generalized and deterministic stochastic petri nets. *Tutorials Notes of the fifth PNPM, Toulouse*, october 1993.

[3] R. David and H. Alla. Du grafcet aux réseaux de petri. *Hermès Science Publications,2 edition*, 2001.

[4] H. Demee G. Bolch, S. Greiner and K. S. Trivedi. Queueing networks and markov chains. *Published by John Wiley and Sons, Hoboken, New Jersey*, 2006.

[5] N. Gharbi. Evaluation des performances et de la fiabilité des systèmes multi-classes avec rappels à l'aide des réseaux de petri stochastiques colorés. *Thèse de Doctorat,Informatique, U.S.T.H.B*, 2007.

[6] M. Gharzouli. "composition des web services sémantiques dans les systèmes peer-to-peer". 2011.

[7] H. Hafi. "protocole pour la sécurité des réseaux sans fil peer to peer". *Université Kasdi Merbah-Ouargla*.

[8] T.W. Hong O. Sandberg I. Clarke, S. G. Miller and B. Wiley. pages 40–49, 2002.

[9] P. Marlier. "sécurité du peer-to-peer". *www.labo-asso.com*, 2007.

[10] M. K. Molloy. Performance analysis using stochastique petri nets. *IEEE Transactions on Computers*, pages 913–917, 1982.

[11] M. K. Molloy. Performance analysis using stochastique petri nets. *IEEE Transactions on Computers*, pages 913–917, 1982.

[12] S. Natkin. Les réseaux de petri stochastiques. *Thèse de Docteur Ingénieur CNAM,Paris*, 1980.

[13] G. Pujolle. "les réseaux". *Eyrolles, Paris, France,1128*, 2008.

[14] C. Ramchandani. Analysis of asynchronous concurrent systems by timed petri nets. *Technical Report, Laboratory of Computer science, MIT, Cambridge*, page 120, 1974.

[15] J. Sifakis. Etude du comportement permanent des réseaux de petri temporisés. *Journées AFCET sur les Réseaux de Petri, Paris, 11*, 1977.