Modélisation et optimisation de réseau infonuagique et de calcul périphérique

par

Njakarison Menja RANDRIAMASINORO

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN CONCENTRATION GÉNIE RÉSEAUX ET
TÉLÉCOMMUNICATION
M. Sc. A.

MONTRÉAL, LE 11 MAI 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Kim Khoa Nguyen, directeur de mémoire
Département de génie électrique, École de Technologie Supérieure

M. Mohamed Cheriet, codirecteur
Département de génie des systèmes, École de Technologie Supérieure

M. Chamseddine Talhi, président du jury
Département de génie logiciel et des TI, École de Technologie Supérieure

M. Aris Leivadeas, membre du jury
Département de génie logiciel et des TI, École de Technologie Supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 23 AVRIL 2021

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Ce travail n'aurait pas pu être réalisé sans les aides et soutiens de mes directeurs, Le Professeur Kim Khoa Nguyen et le Professeur Mohamed Cheriet, je tiens à exprimer mes gratitudes envers eux. Je remercie également tous mes enseignants à l'Ecole Technologie Supérieure qui ont transmis leurs savoir afin de parvenir au terme de ce travail.

Mes remerciements s'adressent aussi aux responsables du Programme Canadien de Bourse pour la Francophonie (PCBF) pour m'avoir offert l'opportunité de poursuivre mes étude auprès d'une des grandes Écoles Canadienne.

J'adresse également ma gratitude envers Mr Benoit Gendron, fondateur et innovateur de société Latence Technologies, de m'avoir accepté comme stagiaire au sein de son entreprise.

Mes derniers remerciements vont à ceux qui m'ont soutenu de près ou de loin. Je pense particulièrement à ma famille et mes proches : Mon père mes frères et sœurs, ma femme ainsi que mon fils, qui m'ont toujours soutenu moralement et financièrement durant mes sséjours ici à Québec.

Je dédie spécialement ce travail à notre défunte MERE, Ravaoharisoa Jeanne Françoise ; sans ses conseils et encouragements, je n'aurais jamais pu continuer.

Modélisation et optimisation de réseau infonuagique et de calcul périphérique

Njakarison Menja RANDRIAMASINORO

RÉSUMÉ

Le calcul périphérique "edge computing" est récemment apparu pour fournir des services aux applications exigeants des contraintes strictes, en termes de qualité de service, telles que les applications 5G à faible latence. Cependant, en raison de leur capacité de calcul limitée, les centres de données "edge" ne sont pas en mesure de répondre à un grand nombre de requêtes des utilisateurs. Une combinaison du "edge" et de l'infonuagique est donc nécessaire pour livrer un service évolutif, à proximité des utilisateurs.

Le défis principal auquel est confronté un système aussi complexe, combinant le "edge", le centre de données "core" et le nuagepublique, est l'allocation optimale des ressources, assurant la qualité de service (QdS) des applications et satisfaisant les contraintes de capacité. Dans ce mémoire, nous présentons un modèle mathématique permettant d'optimiser l'allocation des ressources dans un environnement de "edge cloud", pouvant supporter plusieurs applications.

Les méthodes d'optimisation traditionnelles ne sont plus appropriées pour résoudre un tel problème. Nous modélisons le problème comme un jeu non coopératif et montrons qu'il est transformable à un jeu de transport classique. Nous proposons ensuite une solution pour le jeu basée sur l'élimination des stratégies strictement dominées et la méthode de meilleure réponse. Le but est d'identifier l'équilibre de Nash parmi l'ensemble des points d'équilibres générés dans le jeu. Les simulations expérimentales montrent que notre solution surpasse les travaux précédents.

Mots-clés: Informatique en brouillard, Équilibre de Nash, Jeu de transport, Jeu non coopératif, Meilleure réponse

Modelling and optimizing cloud-edge network

Njakarison Menja RANDRIAMASINORO

ABSTRACT

Edge computing has recently emerged to provide services for applications with strict constraints in terms of quality of service, such as low-latency 5G applications. However, due to their limited computing capacity, edge data centers are not able to serve a large number of user requests. A combination of edge and cloud is therefore required to provide scalable service closer to end users.

The main challenge faced by such a complex system including edge, core data center and public clouds is optimizing resource allocation to guarantee Quality of Service (QoS) requirements and to meet capacity constraints. In this thesis, we present a mathematical model to optimize resources allocation in edge-cloud environment for multiple applications simultaneously.

Traditional optimization methods are no longer appropriate for solving this problem. We model the problem as a non-cooperative game and show that it can be transformed to a classical transportation game. We propose a solution to the game based on the elimination of strictly dominated strategies and the best response method. The goal is to identify the Nash equilibrium among equilibria set generated in the game. Experimental simulations show our solution outperforms the prior work.

Keywords: Edge computing, Nash Equilibrium, Transportation Game, Non-Cooperative game, best response

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ALGORITHMES

Page

LISTE DES SYMBOLES

AP	Application Provider
API	Application Interface
AWS	Amazon Web Service
B2B	Business to Business
BD	Base de données
BRM	Best Response Method
CB	Cloud Broker
CCORAD	Collaborative Computation Offloading and Resource Allocation Optimization
CFFM	Cloud Federation Formation Mechanism
CP	Cloud Provider
CPU	Central Processing Unit
CSB	Cloud Service Brokerage
DC	Data-Center
EC2	Elastic Compute Cloud
ED	Edge Data-center
EU	End-User
GNEP	Generalized Nash Equilibrium Problem
IaaS	Infrastructure as a Service - Infrastructure en tant que service
IDE	Improved Differential Evolution
ILP	Integer Linear Programming
IoT	Internet of Thing
LGNEP	Linear Generalized Nash Equilibrium Problem
MEC	Mobile Edge Cloud
MIP	Mixed Integer Programming
NNCG	N-player Non Cooperative Game
PaaS	Platform as a Service - Plateforme en tant que service
PC	Public Cloud
QdS	Qualité de Service
QoS	Quality of Service

RSDS	Removal of Strictly Dominated Strategy
SaaS	Software as a Service - Application en tant que service
SLA	Service Level Agreement
VM	Virtual Machine

INTRODUCTION

0.1 Contexte

Dans le contexte actuel, les industries se penchent vers une solution plus facile et moins couteuse pour la gestion des affaires. L'une des meilleures solutions est de minimiser l'attache aux services qui peuvent engendrer plus de dépense, par rapport à ce qu'ils rapportent. Il est évident que posséder un système informatique centralisé au sein de son entreprise nécessite de la gestion et de l'entretient du coté matériel et implicitement logiciel. Ces entretiens sont indispensables pour éviter que le système ne tombe en panne , qui pourra engendrer des pertes à l'entreprise, et cela peut être couteuse selon la taille du système. L'adoption d'un système basé sur le Nuage peut éviter ces éventualités.

En effet, l'informatique en nuage est l'une des récentes technologies Internet qui a enregistré une très grande croissance ces 10 dernières années. L'idée initiale, était de centraliser les ressources que les entreprises utilisent pour être accessibles sur n'importe quel terminal connecté. Au fil du temps, cette idée a été exploitée, motivée par l'évolution rapide de l'Internet haut débit. En effet, l'informatique en nuage a rendu plus attrayante la gestion des services et des applications informatiques. C'est l'un des moyens les plus efficaces pour réduire le coût d'investissement sur les matériels et leur gestion. Dans le domaine Infonuagique, les matériels et les applications qui les gèrent sont localisés dans un endroit isolé appelé centre de données (Data Center). La combinaison de ces matériels et applications forme alors le nuage informatique ou l'infonuagique (Jonas et al., 2019).

Dans le paradigme du nuage informatique, il est possible d'être un utilisateur, un fournisseur de services ou les deux. L'utilisation de l'informatique en nuage pourrait être avantageuse et économique. C'est la raison pour laquelle, certaines entreprises sont motivées à devenir fournisseur de service informatique. Cela pour faire plus de bénéfices, accroître les investissements

ou même devenir une plateforme. En plus de ces avantages, l'utilisation d'un tel système offre des opportunités pour les applications telles que : les Applications mobiles interactives, le traitement parallèle par lots, l'analyse de données, les applications bureautiques en ligne, l'Internet de objets, ...

Bien que l'informatique en nuage est une technologie en pleine croissance, son évolution et son utilisation est confrontée à plusieurs obstacles. Ces obstacles sont généralement d'ordre techniques, pour son adoption. Cela peut être la disponibilité des services, le verrouillage des données, la confidentialité et l'auditabilité des données. Des obstacles techniques sur son évolution, tels que le transfert de données, la performance, l'évolutivité du stockage, les erreurs sur les systèmes distribués, l'élasticité peuvent aussi surgir (Jonas *et al.*, 2019).

L'interopérabilité des nuages informatiques est l'une des meilleures solution pour surmonter la plupart de ces obstacles. Rappelons que ce dernier a été conçu sans cette interopérabilité dans l'esprit. Ce principe permettra d'éviter le verrouillage des clients dans un seul infrastructure et de permettre le déploiement de leurs données ou/et applications a travers le nuage. Cela peut résoudre également la disponibilité des services, et permettra de se ressourcer auprès d'autres fournisseurs afin de satisfaire la demande de ses clients durant les heures de pointes où les demandes sont explosifs.

Cette solution d'interopérabilité ne pourra pas résoudre seule les problèmes rencontrés par les fournisseurs d'applications. Plus précisément, certaines applications sur le marché sont actuellement plus exigeantes en terme de qualité de service (QdS) pour leurs bons fonctionnement. La plus sollicité dans les contraintes des applications concerne le temps de réponse aux requêtes. Certaines application exigent une valeur très faible de ce temps, ce que la plupart des fournisseurs ont du mal à offrir. Mettre des ressources à proximité des clients réduit cette difficulté. Ce qui les amènent à adopter l'utilisation des "edge cloud". Ces "edge cloud" offrent une latence faible en étant localisés à leur proximités de leurs utilisations. À cet effet, un architecture basé sur ce

système à proximité parrait une bonne alternative qui pourra améliorer le défis de limitation en latence.Cela fait partie du principal motivation de ce travail de recherche.

0.2 Problématique

L'allocation optimale de ressources virtuelles sur le nuage demeure un problème au niveau des fournisseurs de services. L'origine de ce problème réside sur la façon dont un fournisseur diffuse les requêtes d'allocations des utilisateurs. Ce dernier est paramétré par sa localisation géographique, la quantité de ressource demandée, ainsi que les contraintes de l'application. L'accomplissement de ces exigences valident la satisfaction des clients. Dans ce cas, les fournisseurs devront disposer une solution qui permet de résoudre ce problème d'une façon transparente.

Des chercheurs (Samaan, 2014; Mashayekhy, Nejad & Grosu, 2015; Qiu, Shen & Chen, 2016; Wu, Wu, Li, Zhang, Li & Lau, 2015) ont déjà proposés des solutions pour surmonter ce problème. Ils proposent par contre des approches qui ne tiennent pas en compte le nombre des fournisseurs d'application impliqués, qui peut être multiple. Certains modèles dans ces références ne tiennent pas en compte les contraintes de la QdS, qui est l'un des points importants dans notre travail.

Notre défis dans ce travail est alors de proposer une solution adéquate au problème ci-dessus. Cela pour aidera les fournisseurs d'applications d'identifier d'une façon optimal, où allouer la ressources correspondantes aux demandes des utilisateurs. Le tout en respectant les différentes contraintes imposées par l'application. La solution proposé devra aussi être capable de supporter plus d'un fournisseur d'application. En d'autre terme, nous allons proposer une solution qui pourra assister les fournisseurs d'applications à cartographier l'allocation des ressources, avec un coût minimal, pour ses clients qui sont caractérisés par leur localisation géographiques, tout en tenant en compte les contraintes de l'application.

Pour bien situer notre problématique dans ce travail, on a formulé la question suivante : Selon les exigences des applications en terme de capacité (la taille de la mémoire, le nombre de CPU, la capacité de stockage d'une machine virtuelle ainsi que la bande passante nécessaire, la limite en latence) comment identifier le centre de données capable d'allouer les ressources demandées par tous clients dans une requête, et à moindre coût ?

0.3 Questions de recherche

Les questions de recherches suivantes nous permettront de subdiviser notre problématique :
Question 1 : Pour un fournisseur d'application, il est primordiale de bien choisir où et comment héberger son application sur le nuage avant de se mettre en action. En effet, les exigences de leurs utilisateurs devront être satisfaites. Cela implique alors à bien identifier les centres de données qui vont fournir les ressources et qui répondent aux critères de fonctionnement des applications. Étant donné que plusieurs fournisseurs d'application peuvent être concernés,le modèle du système doit alors supporter cette éventualité. Pour se faire, nous allons poser la question suivante : Pour un moindre coût et tout en respectant les contraintes de l'application ainsi que la multiplicité des fournisseurs, comment les demandes d'allocations en ressources de chaque utilisateur seront elles redistribuées à travers les centres de données répertoriés ? Comment peut-on modéliser ce problème ?

Question 2 : Le problème devient de plus en plus complexe avec l'augmentation du nombre des fournisseurs concernés. Adopter une méthodologie de résolution correspondante à cette problématique s'avère nécessaire. Le modèle est en effet composé d'autant de sous modèles dont chacun associé à une fonction objectif d'un fournisseur qui vise à de minimiser leur coût d'allocation. La question de recherche est correspondante à cette problématique sera comme suit : Comment peut-on satisfaire les différents fournisseurs en même temps, en identifiant l'ensemble ou éventuellement une solution optimale bénéfique pour chacun ?

0.4 Objectifs de la recherche

L'objectif principal de ce travail est de fournir une solution qui permettra de redistribuer d'une façon optimale les ressources aux utilisateurs finaux, selon leurs demande. Les requêtes sont envoyées à l'ensemble des fournisseurs de services qui hébergent les ressources auprès des centres de données. Ces allocations devront en même temps considérer les différentes contraintes imposées.

Cet objectif peut être subdivisé en deux sous objectifs correspondant aux deux questions de recherche définies précédemment :

Objectif 1 : Le premier sous-objectif consiste à établir un modèle d'optimisation pour l'allocation de ressource d'un seul fournisseur d'application. Le modèle tiendra en compte les différentes contraintes liées à l'application et aux fournisseurs de services. Ainsi, il devrait aussi être capable de supporter plusieurs fournisseurs d'application.

Objectif 2 : Le deuxième sous-objectif est de trouver les solutions qui évaluent le choix d'allocation des ressources demandées tout en tenant en compte la multiplicité des fournisseurs.

0.5 Contributions

Une revue de la littérature sur les méthodologies utilisées pour le partage optimale de l'allocation des ressources sur le nuage s'avère un bon départ pour atteindre ces objectifs. Nous allons ensuite à définir notre modèle d'optimisation qui s'appuie sur l'architecture infonuagique basé sur l'utilisation du "edge computing". Cela constitue notre première contribution dans ce mémoire. La résolution de ce problème d'optimisation nous permettra par la suite de bien dispatcher les requêtes des utilisateurs sur un ensemble de centre de données situé sur le "edge", le "core" ou sur le nuage publique. Nous contribuons aussi à améliorer le modèle en ajoutant la prise en charge de multiple fournisseurs d'application. La technique que nous allons utiliser pour cela n'est pas

encore très exploitée dans le domaine infonuagique, surtout pour le partage efficace des ressources virtuelles. Nous allons proposer dans cet effet une solution algorithmique qui effectuera la recherche du point où l'allocation est bénéfique pour tous les acteurs. L'implémentation de cet algorithme dans un environnement expérimental nous permettra de valider l'efficacité de nos démarches et méthodes.

Notre hypothèse dans ce travail est, en envoyant (au système) des requêtes venant de multiples utilisateurs, les ressources à utiliser seront identifiées en un temps record. Le coût d'allocation sera le moindre et que les ressources choisies sont les plus proches de chaque utilisateur.

0.6 Plan du travail

Pour bien mener notre travail, nous allons articuler ce rapport en 3 chapitres. Le premier chapitre est constitué par l'état de l'art. Dedans, nous verrons l'évolution de l'utilisation du nuage informatique depuis sa naissance. Notre revue est basée sur des articles scientifiques qui traitent des problématiques similaires à la notre. Nous allons voir dans ces littératures, le fondement de l'infonuagique, l'avantage de l'utilisation de l'informatique de proximité tel que le brouillard géo-distribués et les "edge". Nous verrons également plusieurs méthodes utilisées par les chercheurs pour optimiser l'allocation de ressources sur le nuage en maximisant leur revenue. Des notions sur la théorie du jeu et le traitement des problèmes qui exploitent cette théorie ainsi que les méthodes utilisées pour la recherche d'équilibre de Nash seront aussi abordés dans cette revue.

Le deuxième chapitre contient la méthodologie que nous allons adopter pour la résolution de notre problématique. Dans ce chapitre, nous allons parler de la méthode d'optimisation linéaire qui pourra résoudre le problème composé d'un seul fournisseur d'application. Nous abordons aussi, la méthode basée sur la théorie de jeux qui est compatible au problème d'optimisation

possédant des multiple fonctions objectives. La solution algorithmique permettant la recherche du point d'équilibre dans un jeu constitue la dernière partie de ce chapitre.

Le troisième chapitre se focalise sur les résultats expérimentaux et leurs interprétations. Nous verrons dans ce chapitre les différentes étapes nous permettant d'analyser et commenter nos résultats. Les configurations de la simulation, les résultats obtenus et leurs interprétations font parties des subdivisions de ce chapitre.

Enfin nous allons conclure notre travail en récapitulant les différentes parties qui évoquent nos contributions. Une proposition pour les travaux futurs complétera ceci et sera discuté dans cette conclusion.

CHAPITRE 1

ÉTAT DE L'ART

Dans ce chapitre, nous allons parcourir les différents travaux de recherche menés dans le domaine infonuagique. Les travaux qui proposent des solutions pour l'allocation efficace de ressources sur le nuage et ceux qui attachent de l'importance sur la qualité de service y seront présentés ici.

1.1 Fédération des Nuages & le "edge cloud"

L'augmentation incessante des demandes d'allocation en ressources est devenue un problème pour les fournisseurs sur le nuage. Dans la plupart des cas, un fournisseur agissant seul aura du mal a satisfaire ces demandes dans la meilleure condition. L'adoption d'une fédération de nuage dont l'objectif est de donner satisfaction aux utilisateurs, en les offrant des ressources qui répondent à leurs ententes pourra remédier à cela. Plusieurs travaux de recherches adoptent l'interconnexion des infrastructures de nuage pour optimiser l'allocation de ressource. Samaan (2014) présente par exemple, un modèle de partage des ressources entre fournisseur en utilisant une fédération.La théorie du jeu est utilisé dans cet article pour évaluer les revenus maximisés obtenus, tout en agissant à l'intérieur ou à l'extérieur d'une fédération. Mashayekhy *et al.* (2015), utilise également une fédération de fournisseur de nuage pour implémenter un mécanisme qui permet aux fournisseurs de former dynamiquement une fédération avec la maximisation du profit. Le jeu de "Hedonic" (Sliwinski & Zick, 2017) est utilisé pour modéliser le mécanisme appelé "Cloud Federation Formation Mechanism" (CFFM). La maximisation du revenu de la fédération est obtenu après soustraction du tarif des instances avec celui du fournisseur multiplié par le nombre des instances à allouer. Le Mécanisme identifie la fédération la plus avantageuse parmi toutes les fédérations formées à l'aide des règles de division "split" et de fusion "merge".

D'ailleurs, la non hétérogénéité des fournisseurs constitue le point faible des modèles présentés dans ces travaux (Samaan, 2014; Mashayekhy *et al.*, 2015). Nous pouvons tout de même nous inspirer des méthodologies de ces auteurs pour établir notre modèle, en apportant notre amélioration. Le problème de minimisation du coût de réservation des ressources est traité par

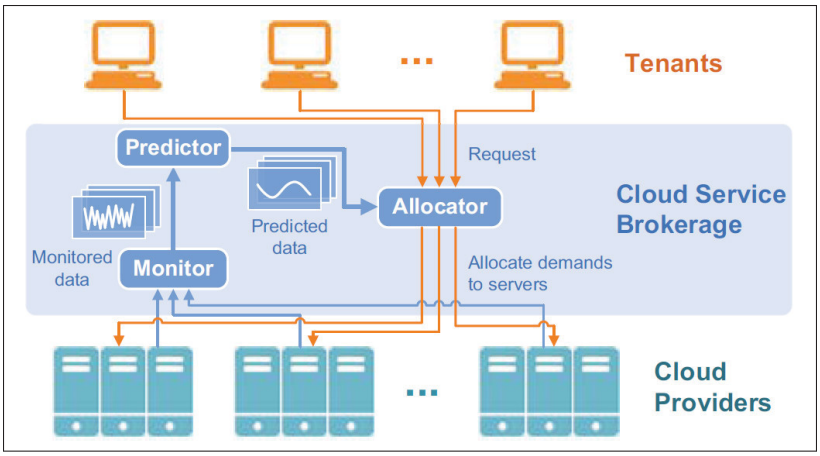


Figure 1.1 Service de courtage sur le nuage
Tiré de Qiu *et al.* (2016)

Qiu *et al.* (2016). Leur système prend en compte l'accord de service (SLA) entre le consommateur et le fournisseur. Le service de courtier "Cloud Service Brokerage" (CSB Fig. 1.1) dans ce système est confronté au défi de la réservation et la répartition des demandes des consommateurs. L'utilisation du courtier sur le nuage (CSB) est l'une des idées qu'on peut prendre en compte dans ce travail.

Allouer des ressources sur des centres de données éparpillés géographiquement est l'objectif de Wu *et al.* (2015). Des politiques standards sont utilisées pour fédérer les fournisseurs de nuages. La méthode utilisée par l'auteur consiste à la migration du contenu pour prévoir les futures demandes, la prédiction des besoins en ressources utilisées pour la décision de migration du contenu. ainsi que l'établissement d'un mécanisme permettant d'ajuster les résultats de l'optimisation. La figure 1.2 représente un architecture géodistribué de nuage informatique présenté par Wu *et al.* (2015).

De leur coté, Di Martino, Cretella & Esposito (2015) adopte, l'interopérabilité et la portabilité via le fournisseur de nuage pour créer leur modèle. L'utilisation de plusieurs fournisseurs nécessite ce paradigme, qui est utilisé pour concevoir la manière dont le fournisseur interagit et partage des ressources avec d'autres afin de répondre aux besoins des services, ou aux demandes des clients. Les rendre interopérables nécessite des services de migration sur les fournisseurs, qui définissent

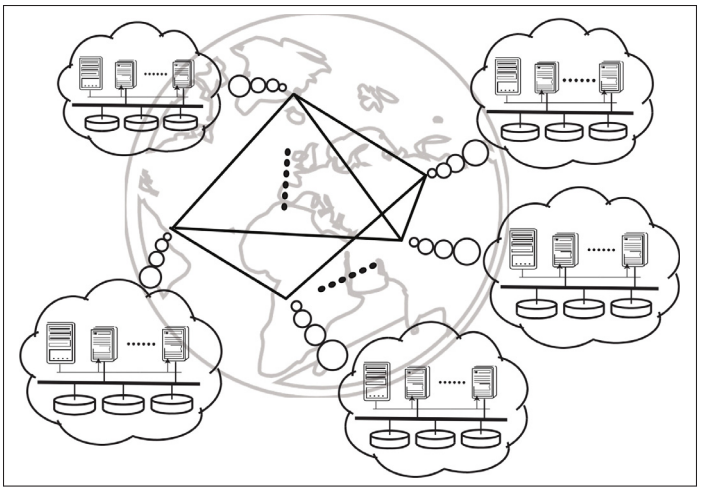


Figure 1.2 Modèle géodistribué de nuage informatique
Tiré de Wu *et al.* (2015)

la portabilité. L'auteur de cet ouvrage (Di Martino *et al.*, 2015) présente la méthodologie et l'approche les plus pertinentes pour résoudre les problèmes d'interopérabilité et de portabilité dans ce domaine.

L'émergence de l'utilisation de l'Internet des objets (IoT) et l'Internet mobile a évoqué plusieurs contraintes au sein du réseau informatique. Ces objets connectés sont actuellement déployés en masse et ont parfois des exigences en terme de temps de réponse. Un serveur d'application éloigné du matériel peut provoquer une latence importante et empêche son bon fonctionnement. De plus, l'augmentation des objets connectés sur le réseau provoque également une congestion de la communication. Cela peut entacher la satisfaction de l'utilisateur, voir même, briser le contrat de service (SLA) avec le fournisseur. Les services pour les utilisateurs peuvent être transférés sur le "edge" et sont exécutés en locales (Hu, Dhelim, Ning & Qiu, 2017). En effet, dans notre conception, la contrainte liée à la latence n'est pas du tout prise à la légère.

1.2 Problème d'allocation optimale de ressources sur le nuage

Le problème d'allocation des ressources sur le nuage repose sur plusieurs facteurs incluant l'emplacement géographique,la limitation de capacité des centres de données, le respect de la QdS, etc. Multiple méthodes sont proposées pour surmonter ce problème. Une approche

collaborative utilisant le “Edge Mobile Computing” et le “Cloud” est présenté par Zhao, Li, Gong & Zhang (2019). Ils proposent un schéma de déchargement collaboratif de calcul et d’optimisation de l’allocation des ressources, ainsi qu’un algorithme de déchargement de calcul distribué et d’allocation de ressources qui permet d’obtenir une solution optimale. Ils utilisent la théorie des jeux pour prendre une décision sur le déchargement des calculs et l’allocation de ressource. Cependant, l’application de leur modèle est limité à un seul domaine d’utilisation. C’est ce que nous allons tenter d’améliore dans notre travail, pour que le modèle supporte multiple fournisseur.

L’utilisation maximale des ressources allouées par l’utilisateur ainsi que la maximisation des revenues des fournisseur de MEC sont les objectifs de Chen, Li, Yang, Nai & Li (2020) dans leur travail. Ils utilisent l’approche du jeu de “Stackelberg” pour la résolution de leur problème. En effet, ils décomposent le problème d’allocation de ressource en multiple sous problème avec un type de ressource unique. Dans ce cas, l’ensemble des solutions d’équilibre des jeux associés aux sous problèmes forment la solution d’équilibre du jeu d’origine. Dans ce travail (Chen *et al.*, 2020), la décomposition du problème en sous problèmes qui sont associés à des sous-jeux est l’avantage de leur approche. Cela réduit considérablement le temps de recherche du point d’équilibre du jeu. Cependant, cette approche n’est pas applicable dans notre cas, sachant que la décomposition affecte la recherche des stratégies optimales des joueurs.

Une méthode intelligente de l’optimisation de l’allocation de machine virtuelle sur le nuage est présenté par Zhang, Zhou & Wang (2020). Leur modèle tient en compte les exigences des utilisateurs ainsi que des fournisseurs. La résolution qu’ils apportent pour résoudre ce problème d’optimisation, développe la méthode d’amélioration de l’évolution différentielle (IDE). L’objectif de leur modèle est aussi de maximiser le revenu des fournisseurs composé d’une fonction multi-objective. Leur modèle tient en compte également la minimisation du coût des ressources pour les utilisateurs, tout en satisfaisant la qualité de service. Le modèle qu’ils développent dans ce travail gère par contre l’allocation de ressources demandées par les utilisateurs directement aux fournisseurs de service.

La résolution du problème d'allocation de ressources dans un MEC en utilisant la méthode généralisé de l'équilibre de Nash est traité par Zaw, Tran, Saad, Han & Hong (2020). Dans leur travail, l'allocation conjointe de la connexion montante et descendante ainsi que l'allocation de ressource sont considérés pour minimiser la latence bout à bout des utilisateurs. L'allocation de ressources est formulé comme un problème généralisé d'équilibre de Nash (GNEP). Ils proposent un algorithme d'allocation de ressources basé sur les pénalités pour résoudre ce GNEP. Cependant, la performance de leur algorithme est impacté par les paramètres de pénalités.

Ces travaux de recherche exposent tous leur point fort dans les méthodologies utilisées par les auteurs . Par contre, ils proposent seulement une solution permettant d'optimiser l'allocation en ressource, sans prendre en compte les contraintes qui peuvent subvenir du coté des utilisateurs. C'est ce que nous allons inclure dans notre modèle en vue d'améliorer les travaux déjà faits.

1.3 Problème de jeu non-coopératif

Notre problème dans ce travail inclus l'utilisation de plusieurs fournisseurs d'applications. Chacun vise à satisfaire les demandes des utilisateurs en fonction de la disponibilité des ressources, auprès des fournisseurs de service et à moindre coût. Cela est vu comme une concurrence entre ces fournisseurs d'applications, qui ne coopèrent pas. À cet effet, un jeu naît entre eux, que nous allons considérer plus tard comme un jeu non-coopératif à plusieurs joueurs. Le jeu est considéré comme un jeu non-coopératif si chaque joueur du jeu agit pour lui-même (Raoof & Al-Raweshidy, 2015). Clempner & Poznyak (2020) traitent ce problème de jeu non-coopératif et définit les concepts de base utilisés dans ce type de jeu. L'auteur de cet article (Clempner & Poznyak, 2020) propose également des solutions sur sur la recherche du point d'équilibre dans un environnement de jeu non-coopératif à N joueurs. Leur méthode nous sera utile pour cette recherche d'équilibre. Le modèle d'allocation et de partage des ressources, est créé par Zafari, Li, Leung, Towsley & Swami (2018) en vue de traiter un problème d'optimisation multi-objectifs. Ils utilisent une stratégie de la théorie des jeux coopératifs pour résoudre le problème. Ils fournissent également les décisions d'allocation de ressources sur le "edge cloud" en satisfaisant d'abord l'application native et en partageant les ressources restantes avec d'autres.

Dans cet article (Zafari *et al.*, 2018), l'auteur adopte l'utilisation d'un environnement "edge" et la théorie du jeu comme méthodologie. Cette méthodologie est similaire à la notre, sauf qu'ils traitent leur problème avec un jeu coopératif.

La modélisation de la compétition entre les fournisseurs de ressources dans l'environnement de "edge cloud" comme un jeu non-coopératif est présentée par Cao, Zhang, Liu & Sheng (2018). Ils considèrent les fournisseurs qui sont composés par des fournisseurs de nuage public et des fournisseurs de "edge cloud". Leur modèle intègre un mécanisme de distribution des tâches auprès des fournisseurs et vise à maximiser le revenu total de ces derniers. Les auteurs de cet article (Cao *et al.*, 2018) démonter également dans leur travail l'existence de l'équilibre de Nash, qui nous sera exploitable dans notre méthodologie. Par contre, comme le précédent article (Zhang *et al.*, 2020), leur système utilise seulement les deux entités, les fournisseurs de service et les utilisateurs, sans les fournisseurs d'application.

La recherche du point d'équilibre de Nash dans un jeu est l'une des étapes la plus fastidieuse. C'est l'identification de ce point qui indique la fin du jeu. Des méthodes sont déjà proposé par des travaux de recherche pour retrouver ce point. Ye, Hu & Lewis (2018) proposent par exemple une méthode pour rechercher le point d'équilibre dans un jeu composé par N-coalition de joueurs non-coopératif. Dans ce travail, les joueurs visent à minimiser leur fonction objective, et la stratégie de recherche d'équilibre de Nash est proposée en fonction de la décision des joueurs dans la coalition. Cette méthode fera partie des idées que nous allons utiliser, vu que l'auteur (Ye *et al.*, 2018) utilise le même type de jeu que le notre.

Un jeu de transport non-coopératif basé sur un problème de transport classique est développé par Stein & Sudermann-Merx (2018). Ce travail aborde le problème de la preuve de l'existence et de la recherche de l'équilibre de Nash pour ce genre de jeu. Ils utilisent dans cet article, l'algorithme "Sign Bingo" pour évaluer l'équilibre du problème et l'efficacité de leur méthode. Notre travail présente une similitude étroite avec cet article sur la manière où nous considérons notre système comme un jeu non-coopératif. Notre objectif est par contre de trouver le point d'équilibre où les fournisseurs optimiseront la tarification de leurs ressources.

CHAPITRE 2

MÉTHODOLOGIE

2.1 Introduction

Dans ce chapitre, nous allons discuter des méthodologies à utiliser pour traiter notre problématique. Notre problème concerne l'optimisation de l'allocation de ressources sur un nuage hétérogène géodistribué. Sa formulation se base sur les informations correspondantes aux fournisseurs de ressources ainsi qu'aux fournisseurs d'application.

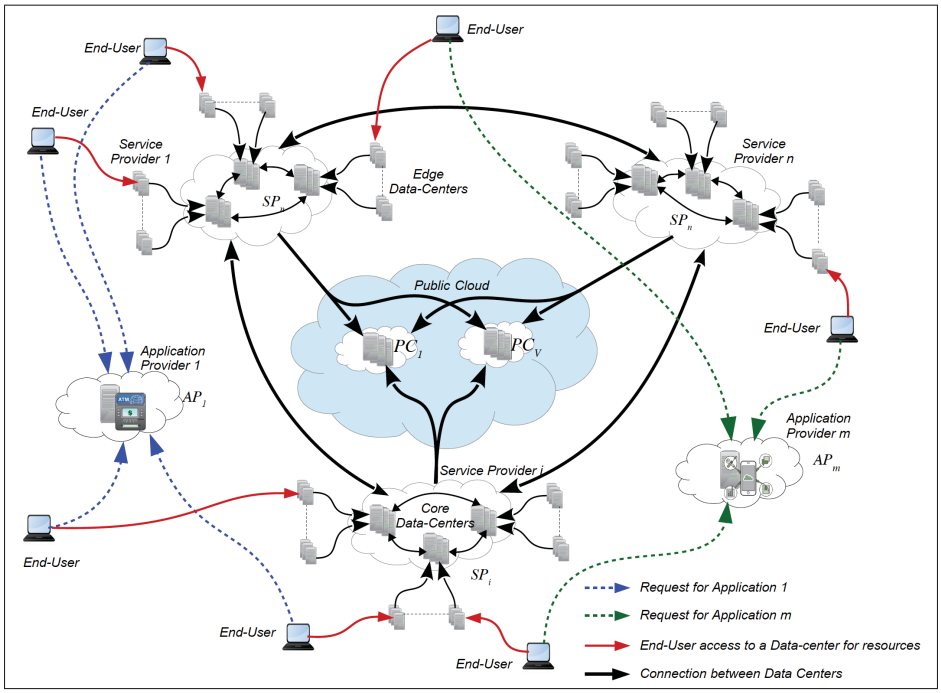


Figure 2.1 Architecture général du système

La figure 2.1 représente le système en générale, qui est composé d'autant de fournisseurs d'application que de fournisseurs de service.

2.2 Description du système

L'ensemble de notre système est composé par 3 catégories d'acteurs dont :

- les fournisseurs de service,
- les fournisseurs d'application,
- les utilisateurs finaux.

Les fournisseurs de services sont ceux qui mettent en location les ressources virtuelles sur le nuage. Il peuvent être des fournisseurs de nuage publique privé ou hybride. Un fournisseur de service a en sa possession la quantité minimum de ressources éligibles aux critères des applications. Les fournisseurs d'application sont ceux qui offrent leurs applications aux clients (comme exemple, les fournisseurs de service de télécommunication, fournisseurs de streaming vidéo,...). Ces 2 types de fournisseur se coopèrent pour fournir des services aux utilisateurs finaux.

2.2.1 Les fournisseurs de service

Nous nous intéressons au problème d'allocation optimale des ressources dans un système hétérogène. Un tel système incluse les centres de données, qui sont interconnectés sous forme d'une structure hiérarchique. Ce modèle vise à fournir une faible latence qui est une métrique de la QdS difficile à contrôler, même avec une bonne bande passante. L'adoption du "edge Computing" est l'une des solutions qui pourra résoudre ce problème, surtout sur l'utilisation des applications interactives et en temps réel (Satyanarayanan, Bahl, Caceres & Davies, 2014).

Pour notre cas, les micro-centres de données géographiquement situés à proximité des utilisateurs finaux seront exploités. Nous les considérerons comme des brouillards géodistribués. Utiliser les ressources à proximité de ces utilisateurs apporte des avantages en termes de temps de réponse, ce qui est particulièrement nécessaire pour les applications du réseau 5G à venir. Cela permettra également de réduire la consommation d'énergie (Shi, Cao, Zhang, Li & Xu, 2016).

La hiérarchie des centres de données est constituée de trois niveaux qui regroupent les "edge data-centers", les Noyaux ou "core data-centers" et les fournisseurs de nuage publique. Les "edge data-centers" sont situés à proximité des utilisateurs afin de réduire la latence du service durant l'exécution des applications. Ils constituent les éléments sur le premier niveau de la

hiérarchie et sont souvent de faible capacité. Les "edge data-centers" sont ensuite connectés à des "core data-centers", un peu plus éloignés des utilisateurs (appartenant au même entreprise de télécommunication agissant comme fournisseur de nuage privé). Ces derniers forment alors le deuxième niveau de la hiérarchie.

Ces "core data-centers" peuvent être connectés à des nuages publiques (comme Google cloud, Microsoft Azure ou Amazon AWS). Ces fournisseurs de nuage publiques forment le troisième niveau. Sur la base de cette vue architecturale, nous visons à minimiser le coût d'allocation de ressource, tout en entretenant un niveau de QdS acceptable. Cela pourra se faire en résolvant un modèle d'optimisation qui intègre les configurations des centres de données, les contraintes de capacité ainsi que les exigences de la QdS.

2.2.2 Les fournisseurs d'application

Les fournisseurs d'application sont ceux qui offrent directement leurs service/application aux utilisateurs finaux. Ces utilisateurs peuvent être des personnes physiques, des entreprises ou organisations, ou même d'autres fournisseurs. Un fournisseur d'application fonctionne comme une passerelle entre les fournisseurs de services et les utilisateurs. Les applications offertes seront déployées sur le nuage, utilisant des ressources virtuelles des centres de données. Chaque application a ses critères en terme de QdS, comme la latence minimale acceptée par l'application et la bande passante pour la faire fonctionner. De ce fait, une ressource virtuelle qui ne remplisse ces critères ne sera pas sélectionnée. En effet, selon l'application offerte et ses critères de fonctionnement, la tâche du fournisseur d'application est de trouver la ressource adéquate pour chaque demande d'utilisateur. L'emplacement de la ressource sur le plan géographique joue alors un rôle important, sachant que la latence dépend principalement de la distance entre le serveur et l'utilisateur.

2.3 Modélisation du système

Pour atteindre notre objectif, qui est d'optimiser l'allocation des ressources sur le nuage, en minimisant le coût de chaque allocation de ressources, nous allons modéliser le système en 2 étapes. Sachant que le système est composé de plusieurs fournisseurs d'application, et que c'est au niveau de ces types de fournisseur que le problème se pose, il est alors judicieux de produire un modèle qui généralise l'optimisation de l'allocation de ressource pour les fournisseurs. C'est la première étape de notre travail, qui consiste à formuler le modèle comme un problème d'optimisation linéaire avec une unique fonction objective.

2.3.1 Modèle linéaire de l'allocation de ressources

L'objectif de chaque fournisseur d'application dans le nuage est de maximiser leurs profit, ce qui implique la minimisation du coût des ressources à louer. Il est alors plus simple de considérer un seul fournisseurs servant plusieurs utilisateurs, étant donné que le modèle prendra la même forme pour les autres. Chaque requête d'utilisateur demandant à utiliser une application implique une allocation de ressource virtuelle sur le nuage (sous forme de machine virtuelle). Une application ne pourra pas fonctionner que si critères suivantes sont respectées :

la capacité d'une machine virtuelle :

- le nombre des CPU C_m,
- la taille de la mémoire vive M_m,
- la taille du disque stockage S_m,

les métriques de la qualité de service :

- latence entre l'utilisateur et le centre de donnée hébergeant la machine virtuelle τ_m,
- la bande passante de la connexion bw_m.

Il faut noter que le coût total de l'allocation des ressource dépend : des caractéristiques des ressources, du fournisseur de service qui l'offre, du coût de la connexion.

$$RC_{tot} = \sum_{K.\in\{SP_n\}} Rc_K + Lc_{Eu-SP_n} \tag{2.1}$$

2.3.2 Formulation du modèle

La formulation de ce problème consiste alors a minimiser le coût total de l'allocation en ressources décrit par l'équation (2.1). Supposons que la variable de décision de l'allocation de ressources est x, nous avons alors l'équation suivante comme fonction objective de notre modèle :

$$Min \quad RC_{tot} = \sum_{k\in\{SP_n\}} x_k * \left(Rc_k + Lc_{Eu-SP_n}\right) \tag{2.2}$$

L'allocation des ressources auprès des différents niveau des centres de données utilisent chacun une variable de décision différente. Sur le "edge" cette variable de décision est désignée par x, sur le "core", la variable est y et sur le nuage public cette variable est z.

Ce modèle est présenté par l'équation (2.3)

$$\begin{aligned} Min \qquad \sum_{u\in\{EU_u^m\}} \sum_{n=1}^{N} \Bigg(\sum_{p(n)=1}^{P(n)} x_{m|u}^{n|p(n)} * P_n^{p(n)}(VM_m) + \sum_{l=1}^{N} y_{m|u}^{n|l} * P_l(VM_m) \\ + \sum_{v=1}^{V} z_{m|u}^{n|v} * \left(P_v(VM_m) + Lc_n^v\right) \Bigg) \end{aligned} \tag{2.3}$$

Les différentes notations utilisés dans toutes les équations sont décrites dans le tableau 2.1 (page 20).

Tableau 2.1 Liste des symboles et notations

Symboles	**Description**
$\{AP_m\}$	Ensemble des M fournisseurs d'application $\{AP_1, ..., AP_M\}$
$\{EU^u_m\}$	Ensemble d'utilisateurs finaux utilisant une application m de AP_m
$\{SP_n\}$	Ensemble N des fournisseurs de service $\{SP_1, ..., SP_N\}$
$\{ED^n_{p(n)}\}$	Ensemble des centres de données "edge" d'un fournisseur de service $\{SP_n\}$
$\{PC_v\}$	Ensemble de V nuage publique $\{PC_1, ..., PC_V\}$
m, M	Indice de l'application, nombre des fournisseurs d'application ($m = 1, ..., M$)
n, N	Nombre des fournisseurs de service ($n = 1, ..., N$)
$p(n), P(n)$	Nombre des centres de données "edge" du fournisseur SP_n
v, V	Nombre des nuages publiques ($v = 1, ..., V$)
τ_m	Latence maximale pour pouvoir utiliser l'application m de AP_m
U_m	Nombre total des utilisateurs de l'application m, ($u_m = 1, ..., U_m$)
$\mathcal{L}(A\|B)$	Latence mesurée entre deux nœuds A et B
$\mathcal{L}(EU^u_m\|ED^n_{p(n)})$	Latence mesurée entre l'utilisateur EU^u_m et le "edge" $ED^n_{p(n)}$
$\mathcal{L}(ED^n_{p(n)}\|SP_n)$	Latence mesurée entre le "edge" $ED^n_{p(n)}$ et le fournisseur de service SP_n
$\mathcal{L}(SP_n\|SP_l)$	Latence mesurée entre le fournisseur SP_n et un autre SP_l ($l = 1, .., N$)
$\mathcal{L}(SP_n\|PC_v)$	Latence mesurée entre fournisseur SP_n et le nuage publique PC_v
$Co^n_{p(n)},\ Co_n$	Nombre de CPU disponible sur $ED^n_{p(n)}$ respectivement SP_n
$Me^n_{p(n)},\ Me_n$	Mémoire totale disponible sur $ED^n_{p(n)}$ respectivement SP_n
$St^n_{p(n)},\ St_n$	Stockage totale disponible sur $ED^n_{p(n)}$ respectivement SP_n
$P^{p(n)}_n(VM_m)$	Coût d'allocation d'une VM_m correspondante à l'application m sur $ED^n_{p(n)}$
$P_n(VM_m)$	Coût d'allocation d'une VM_m correspondante à l'application m sur SP_n
$P_v(VM_m)$	Coût d'allocation d'une VM_m correspondante à l'application m sur PC_v
C_m, M_m, S_m	CPU, mémoire et Stockage requis pour une application m
VM_m	Machine virtuelle possédant M_m mémoire, C_m CPU, S_m stockage
bw_m	Bande passante requise pour lancer une application du fournisseur AP_m
$Co^\lambda, Me^\lambda, St^\lambda$	CPU, Mémoire, Stockage disponible auprès de SP_n ou $ED^n_{p(n)}$
Bw_n	Bande passante disponible pour la connexion internet auprès de SP_n
Lc^v_n	Coût du lien de la connexion de SP_n vers PC_v
$x^{\{\alpha,\beta,\gamma\}}_{m\|u},\ x^\lambda_{m\|u}$	Variable indicateur pour l'utilisateur u de l'application m
$\{\mathcal{X}_m\}$	Ensemble des décisions (stratégie) de AP_m ($\{\{\mathcal{X}^1_m\}, .., \{\mathcal{X}^\Lambda_m\}\}$)
$\{\mathcal{X}^*_m\} = \mathcal{X}^{*\lambda}_m$	Stratégie de AP_m sur le point d'équilibre ($\mathcal{X}^{*\lambda}_m = \{x^{*\lambda}_{m\|1}, ..., x^{*\lambda}_{m\|U_m}\}$)
$\{\alpha, \beta, \gamma\}$	indices de la variable x (selon le centre de données source de la VM_m)
δ	Indice de la variable x $\delta \in \{\alpha, \beta, \gamma\}$ et $\delta \in [1, .., \Delta]$
RC_m	Coût total des ressources pour le fournisseur d'application AP_m
Rc_m, Lc_m	Coût d'une ressources respectivement du lien pour l'application m

Dans cette équation (2.3), le $P_n^{p(n)}(VM_m)$ représente le coût d'allocation de la machine virtuelle (VM), correspondante à une application sur le "edge". $P_l(VM_m)$ le coût de la même VM sur le "core" et $P_v(VM_m)$ sur le nuage publique, tandis que Lc_n^v, représente le coût de la connexion du lien entre le nuage publique et le l'utilisateur. Et comme nous le constatons, l'objectif est de minimiser ce coût total d'allocation pour une requête groupée, demandant au fournisseur de lancer une application.

Ce modèle est effectivement associé aux contraintes suivantes :

- **les contraintes liées à la latence :**

$$x_{m|u}^{n|p(n)} * \mathcal{L}\left(EU_m^u | ED_{p(n)}^n\right) \leqslant \tau_m \tag{2.4}$$

$$y_{n|l}^{\beta} * \left(\mathcal{L}\left(EU_m^u | ED_{p(n)}^n\right) + \mathcal{L}\left(ED_{p(n)}^n | SP_n\right) + \mathcal{L}\left(SP_n | SP_l\right)\right) \leqslant \tau_m \tag{2.5}$$

$$z_{n|v}^{\gamma} * \left(\mathcal{L}\left(EU_m^u | ED_{p(n)}^n\right) + \mathcal{L}\left(ED_{p(n)}^n | SP_n\right) + \mathcal{L}\left(SP_n | PC_v\right)\right) \leqslant \tau_m \tag{2.6}$$

- **contraintes liées aux capacités des centres de données :**

Ces contraintes sont composés par le nombre de CPU (C) la Mémoire (M) et le stockage (S)

$$\sum_{m=1}^{M} \sum_{u \in \{EU_u^m\}} x_{m|u}^{n|p(n)} * M_m \leqslant Me_{p(n)}^n \quad \forall p(n) \in \{ED_{p(n)}^n\} \tag{2.7}$$

$$\sum_{m=1}^{M} \sum_{u \in \{EU_u^m\}} \sum_{l=1}^{N} y_{m|u}^{n|l} * M_m \leqslant Me_n \quad \forall l \in \{SP_n\} \tag{2.8}$$

Ces contraintes sont les mêmes avec $C_m(\leqslant Co)$ et $S_m(\leqslant St)$.

Enfin, la contrainte de la bande passante est la suivante :

$$\sum_{m=1}^{M} \sum_{u \in \{EU_u^m\}} z_{m|u}^{n|v} * \beta_m \leqslant Bw_n \quad \forall l \in \{SP_n\} \tag{2.9}$$

$$\sum_{n=1}^{N}\Big(\sum_{p(n)=1}^{P(n)} x_{m|u}^{n|p(n)} + \sum_{l=1}^{N} y_{m|u}^{n|l} + \sum_{v=1}^{V} z_{m|u}^{n|v}\Big) = q_u^m \quad \forall u = 1,..,U \quad \forall m = 1,..,M \tag{2.10}$$

$$x_{m|u}^{n|p(n)}, y_{m|u}^{n|l}, z_{m|u}^{n|v} \in [0,1] \quad \text{des nombres entiers} \tag{2.11}$$

$$\forall u = 1,..,U \quad m = 1,..,M \quad n = 1,..,N \quad p(n) = 1,..,P(n) \quad and \quad v = 1,..,V$$

En vue de faciliter son implémentation, le modèle ci-dessus peut être converti en utilisant une seule variable de décision. Notons cette variable par x. La fonction objective dans l'équation (2.3) devient alors :

$$Min\ RC_m = \sum_{u\in\{EU_u^m\}} \sum_{n=1}^{N}\Big(\sum_{p(n)=1}^{P(n)} x_{m|u}^{\alpha} * P_n^{p(n)}(VM_m) + \sum_{l=1}^{N} x_{m|u}^{\beta} * P_l(VM_m) + \sum_{v=1}^{V} x_{m|u}^{\gamma} * \left(P_v(VM_m) + Lc_n^v\right)\Big) \tag{2.12}$$

où :

$$\alpha = \big(M * (u-1) + m - 1\big) * \sum_{i=1}^{N} P(i) + \sum_{j=1}^{p(n)} P(j) - P(n) + p(n) \tag{2.13}$$

$$\beta = \big(M * (u-1) + m - 1\big) * \sum_{i=1}^{N} N + \sum_{j=1}^{n} N - N + l \tag{2.14}$$

$$\gamma = \big(M * (u-1) + m - 1\big) * \sum_{i=1}^{N} V + \sum_{j=1}^{n} V - V + v \tag{2.15}$$

$$\forall u = 1,..,U; \quad m = 1,..,M; \quad n = 1,..,N; \quad p(n) = 1,..,P(n) \quad and \quad v = 1,..,V$$

En utilisant les équations (2.1) et (2.12), on peut obtenir une forme encore plus simple du modèle :

$$Min\ RC_m = \sum_{\{EU_u^m\}} \sum_{\{AP_m\}} \sum_{\{SP_\delta\}} x_{m|u}^{\delta} * \Big(Rc_m(SP_\delta) + Lc_m(SP_\delta)\Big) \tag{2.16}$$

Et les contraintes correspondantes sont les suivantes :

$$x^{\delta}_{m|u} * \mathcal{L}\left(EU^{u}_{m}|SP_{\delta}\right) \leqslant \tau_{m} \quad \forall u = 1, .., U_{m} \tag{2.17}$$

$$\sum_{m=1}^{M} \sum_{u \in \{EU^{m}_{u}\}} x^{\delta}_{m|u} * M_{m} \leqslant Me^{\delta} (\text{ les mêmes pour } C_{m}, S_{m}, bw_{m}) \tag{2.18}$$

$$\sum_{\{EU^{m}_{u}\}} \sum_{\{SP_{\delta}\}} x^{\delta}_{m|u} = U_{m} \tag{2.19}$$

$$x^{\delta}_{m|u} \in [0, 1] \tag{2.20}$$

$\delta \in \{\alpha, \beta, \gamma\}$, définis dans les équations (2.13) - (2.15).

L'implémentation de ce modèle nous permet de résoudre l'optimisation du coût d'allocation des ressources. En effet, pour que le système peut supporter multiple fournisseur d'application, chaque fournisseur a son propre modèle. Ce qui est avantageux dans ce problème est que, le modèle associé à chaque fournisseur a une forme identique. Nous somme face à un problème dont le nombre de fonction objective est égale au nombre des fournisseurs impliqués. Ce qui nous amène dans la partie suivante qui vise à modéliser un problème à multiple fonction objective.

2.4 Méthodologie basée sur la théorie du jeu

À présent nous avons un modèle qui est applicable sur plus d'un fournisseur d'application. La méthode de résolution utilisant la programmation linéaire nous permet tout même de résoudre ce problème mais avec une efficacité faible. En effet, notre problème génère plusieurs résultats optimales, provenant de chaque fonctions objectives associées aux fournisseurs d'applications. Pour surmonter à cela, nous allons adopter la méthode basée sur la théorie des jeux.

2.4.1 La théorie des jeux

Par définition, "La théorie des jeux est un concept mathématique qui traite la formulation de la stratégie correcte qui permettra à un individu ou une entité (c'est-à-dire un joueur), confronté à un défi complexe, de réussir à relever ce défi" (Raoof & Al-Raweshidy, 2015). Selon toujours ce même auteur, toute entreprise peut être considérée comme un jeu joué contre des concurrents, voire contre des clients. Dans la théorie des jeux, l'objectif de chaque joueur est de gagner plus à chaque fois qu'il entreprend une action, tout en considérant la future action de ses adversaires. Il existe plusieurs classifications de jeux qu'on peut rencontrer selon son domaine d'application :

- **les jeux de coalition (jeu coopératif et non-coopératif) :** un jeu est coopératif si les joueurs ont la possibilité de former une coalition leur permettant d'améliorer leurs futures décisions d'actions. Par contre, un jeu est non coopératif si cette coalition ne pourra pas exister entre les joueurs et chaque joueur agit pour son propre bien,
- **les jeux de stratégies :** les joueurs dans ces jeux prennent leurs décisions simultanément au début du jeu. Un joueur ne possède aucune information sur l'action de son adversaire (Raoof & Al-Raweshidy, 2015),
- **jeu à somme nulle :** un jeu est à somme nulle s'il existe une balance entre le gain ou la perte de chaque joueur. En effet, la somme des revenues des joueurs est toujours nulle.

Ce ne sont que quelque types de jeu, qui sont les plus populaires, mais il existe une multitude de type de jeu. Ces différents types de jeux sont utilisé pour résoudre des problème complexes dans certains domaines tels que la modélisation mathématique, la science de l'information, le réseau, le transport, l'économie, les affaires, la gestion de projet, la biologie, la science politique, etc.

2.4.1.1 Composition d'un jeu

Un jeu est généralement composé par les triplets suivants :

- **les joueurs** (Jo) **:** c'est l'entité qui entreprend les actions, récolte les gains selon ses actions ou stratégies.

- **la stratégie** (St) **:** définit l'ensemble des actions ou de décision que peut prendre un joueur dans un jeu.
- **le gain** (Re) **:** c'est la récompense ou revenue qu'obtient un jouer selon son choix de décision, ou stratégie.

Nous pouvons alors formuler un jeu spécifique que nous notons par $\mathcal{G}$, par la représentation suivante :

$$\mathcal{G} = \langle Jo, St, Re \rangle \tag{2.21}$$

2.4.1.2 L'équilibre de Nash dans un jeu

Selon la définition présenté par Stein & Sudermann-Merx (2018), l'équilibre de Nash dans la théorie des jeux est un terme qui défini la situation dans laquelle les stratégies de chaque participants dans le jeu sont optimales. Aucun joueur ne souhaite pas prendre une décision unilatérale et de changer de stratégie à ce point d'équilibre pour ne pas subir une perte ou la risque de diminuer son gain. L'équilibre de Nash regroupe alors la stratégie optimale de chaque joueur.

Considérons par exemple un ensemble de N joueurs $Jo = \{Jo_1, .., Jo_N\}$ qui ont respectivement l'ensemble des stratégies $\{\{St_1\}, .., \{St_N\}\}$ correspondantes aux revenues $\{\{Re_1\}, .., \{Re_N\}\}$.

Pour la m ième stratégie du joueur n noté par St_n^m, les stratégies des autres joueurs sont définies par $St_n^{-m} = \{St_1^{-m}, .., St_{n-1}^{-m}, St_{n+1}^{-m}, ..., St_N^{-m}\}$. Notons aussi l'ensemble des stratégies des joueurs en équilibre de Nash par :

$$NE \Rightarrow \{St^*\} = \{St_1^{*i}, .., St_n^{*j}, ...St_N^{*k}\} \tag{2.22}$$

L'ensemble est formé par la i^{eme} stratégie du joueur numéro 1, ..., j^{eme} stratégie du n^{eme} joueur, ..., k^{eme} stratégie du jouer numéro N.

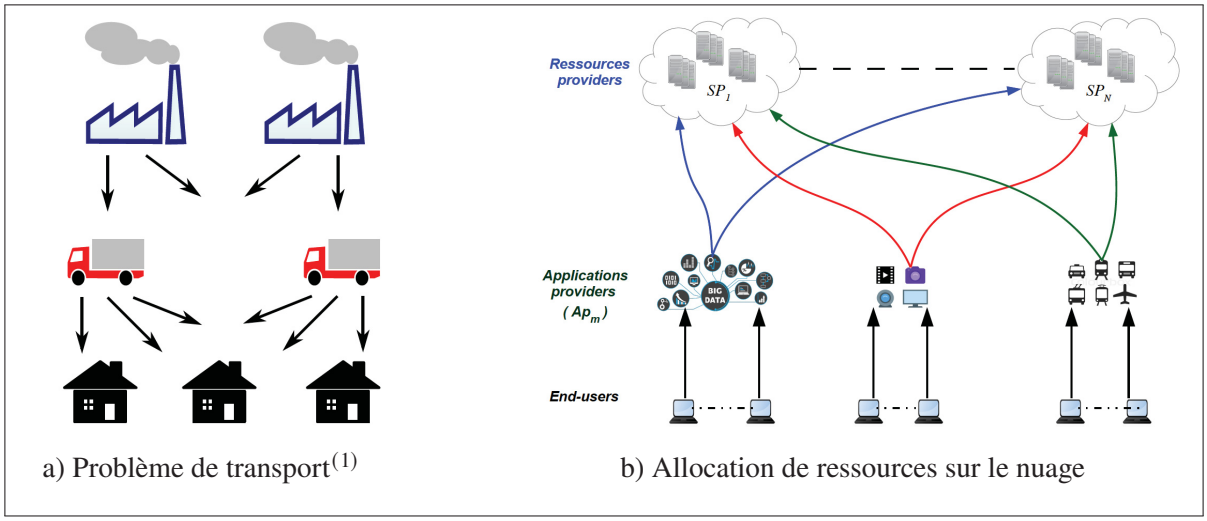
L'ensemble des stratégies de l'équation (2.22) est un équilibre de Nash si et seulement si :

$$\Leftrightarrow Re_n\left(St_n^{*j}, \{St_n^{-j}\}\right) \geqslant Re_n\left(St_n^{l}, \{St_n^{-l}\}\right) \quad \forall St_n^{*j} \text{ and } St_n^{l} \in \{St_n\}, n = 1, .., N \qquad (2.23)$$

Ce qui veut dire que le revenue du joueur n (Re_n) obtenu en adoptant la stratégie du point d'équilibre (St_n^{*j}) considérant les stratégies des autres joueurs $(St_n^{-j}) \Rightarrow Re_n\left(St_n^{*j}, \{St_n^{-j}\}\right)$ est toujours avantageux par rapport aux autres revenus obtenus en adoptant d'autres stratégies différentes (St_n^{l}) considérant les stratégies des autres joueurs $(St_n^{-l}) \Rightarrow Re_n\left(St_n^{l}, \{St_n^{-l}\}\right)$

2.4.2 Jeu de transport classique

Nous nous intéressons particulièrement à ce type de jeu classique, vu sa similarité avec notre problème. Dans leur travail, Stein & Sudermann-Merx (2018) modélise le même problème dans le domaine du transport, qui regroupe plusieurs transporteurs compétitifs dans un jeu classique. Les joueurs de ce jeu sont composés par des transporteurs qui vise à maximiser leurs revenus en transportant les marchandises des fabricants vers les clients.



a) Problème de transport[1] b) Allocation de ressources sur le nuage

Figure 2.2 Comparaison avec le problème de transport
[1]Tiré de Stein & Sudermann-Merx (2018)

De la même manière, nous considérerons aussi notre système comme un jeu classique basé sur un problème de transport. La figure 2.2 illustre la similarité entre ces deux problèmes. Pour notre cas,

les joueurs sont formés par des fournisseurs d'applications qui louent des ressources auprès des fournisseurs de service nuage. Ils donnent ensuite l'accès à ces ressources aux utilisateurs finaux pour l'hébergement de leurs applications. Une compétition entre ces fournisseurs d'application se forme alors pour allouer les ressources nécessaires pour leurs clients. Ainsi, aucune coopération n'existe entre ces fournisseurs. Nous avons ici alors, un jeu non-coopératif qui ressemble au problème de transport. Notre objectif final est de trouver le point d'équilibre du jeu où les joueurs (les fournisseurs d'application) seront satisfaits sur l'utilisation des services.

2.5 Modélisation par le jeu non-coopératif à N-joueurs

Dans cette partie, nous allons utiliser notre modèle défini dans la section 2.3.2 et la représentation d'un jeu dans l'équation (2.21) pour formuler notre modèle. Nous le considérons comme un jeu non-coopératif à N-joueurs (NCNG) composé d'un ensemble M de fournisseurs d'application $\{AP_m\}\forall m = 1, .., M$. Il y a donc compétition entre ces joueurs, AP_m qui composent le jeu. Chacun d'entre eux souhaite allouer des ressources virtuelles à partir d'un ensemble de fournisseurs de ressources $\{SP_\delta\}\ \forall\delta \in \{\alpha, \beta, \gamma\}$, en fonction des requêtes des utilisateurs finaux $\{EU_u^m\}\ \forall u = 1, .., U_m$ et $m = 1, .., M$ ($\alpha, \beta,\ et\ \gamma$ sont définis dans les équations 2.13 à 2.15).

Ici, chaque centre de données est considéré comme fournisseur de ressources. Soit $x_{m|u}^{\delta}$, $\delta \in \{\alpha, \beta, \gamma\}$, le nombre de ressources demandées par un utilisateur u pour une application appartenant à AP_m et hébergé par le fournisseur de ressources $\{SP_\delta\}\ \forall\delta \in \{\alpha, \beta, \gamma\}$.

Comme défini dans l'équation (2.16), chaque AP_m dans le système vise à minimiser leurs frais d'allocation de ressources. Chaque AP_m est alors associé au modèle suivant :

$$Min\ RC_m = \sum_{\{EU_u^m\}} \sum_{\{AP_m\}} x_{m|u}^{\delta} * \Big(Rc_m(SP_\delta) + Lc_m(SP_\delta)\Big) \tag{2.24}$$

Dans cette équation, $Rc_m(SP_\delta)$ et $Lc_m(SP_\delta)$ définissent le coût des ressources pour l'application m sur SP_δ, respectivement, le coût de la connexion pour accéder aux ressources hébergées sur

SP_δ. Chaque fournisseur d'application partage des contraintes identiques :

$$x^{\delta}_{m|u} * \mathcal{L}\left(EU^{u}_{m}|SP_\delta\right) \leqslant \tau_m \quad \forall u = 1, .., U_m \tag{2.25}$$

$$\sum_{m=1}^{M} \sum_{u \in \{EU^{m}_{u}\}} x^{\delta}_{m|u} * M_m \leqslant Me^{\delta} \tag{2.26}$$

$$\sum_{m=1}^{M} \sum_{u \in \{EU^{m}_{u}\}} x^{\delta}_{m|u} * C_m \leqslant Co^{\delta} \tag{2.27}$$

$$\sum_{m=1}^{M} \sum_{u \in \{EU^{m}_{u}\}} x^{\delta}_{m|u} * S_m \leqslant St^{\delta} \tag{2.28}$$

$$\sum_{m=1}^{M} \sum_{u \in \{EU^{m}_{u}\}} x^{\delta}_{m|u} * bw_m \leqslant BW^{\delta} \tag{2.29}$$

$$\forall \delta \in \{\alpha, \beta, \gamma\}$$

2.5.1 Modèle du jeu

Selon Clempner & Poznyak (2020), un jeu composé de N-joueurs où les joueurs sont associés à un ensemble fini de stratégie pure et correspondante à un ensemble de gain, forment un jeu fini.

Notons par :

$$\{\mathcal{X}_m\} = \{\{\mathcal{X}^1_m\}, \{\mathcal{X}^2_m\}, ..., \{\mathcal{X}^\lambda_m\}, ..., \{\mathcal{X}^\Lambda_m\}\} \tag{2.30}$$

$$= \{\{x^1_{m|1}, ..., x^1_{m|U_m}\}, ..., \{x^\lambda_{m|1}, ..., x^\lambda_{m|U_m}\}, ..., \{x^\Lambda_{m|1}, .., x^\Lambda_{m|U_m}\}\} \tag{2.31}$$

l'ensemble des décisions d'un AP_m dans une requête.
Avec $|\Lambda|$ = nombre total des stratégies pour le joueur m ($\forall m = 1..M$) et Um le nombre total des utilisateurs qui envoient des requêtes au joueur AP_m.

Le triplet qui compose le jeu non-coopératif à N-joueurs dans notre cas sera formulé comme suit :

$$\mathcal{G} = \left\langle AP_m, \mathcal{X}_m, RC_m \right\rangle \quad \forall m = 1, ..., M \tag{2.32}$$

Dans cette formulation :

- AP_m désigne le joueur m.
- $\mathcal{X}_m$ est la stratégie du joueur définie par un ensemble de décision du joueur m.
- RC_m la fonction objectif qui vise à minimiser le coût des ressources demandées dans la requête.

Dans ce modèle nous pouvons remarquer que les stratégies des joueurs (fournisseurs d'application) sont composées par des ensembles finis de décision qui sont associées à des ensembles finis de coût de ressources. Se référant à la définition de Stein & Sudermann-Merx (2018), nous pouvons considérer notre jeux comme un jeu fini. Quelque règlement sont défini en cas de conflit entre l'attribution des ressources à des requêtes.

En effet,un utilisateur qui a envoyé une requête pour l'utilisation d'une ressource sera redirigé vers le serveur qui lui est plus proche, tant que ce dernier dispose au moins une VM ayant la capacité permettant héberger l'application en question. En cas d'insuffisance en capacité sur les "edge", le "core data-center" sera sollicité à son tour. Le nuage publique est le dernier recours dans le cas où l'ensemble des fournisseurs locaux ne peuvent plus subvenir aux requêtes soumises.

2.5.2 Recherche de l'équilibre de Nash (EN)

2.5.2.1 Existence de l'EN

L'équilibre de Nash constitue le point où les stratégies des joueurs sont des stratégies optimales selon la définition de la précédente section. Notre objectif dans la résolution de notre modèle est d'identifier ce point optimal, où chaque joueur dans le jeu trouvent leurs avantages. Il nous est alors nécessaire d'identifier ce point d'équilibre de Nash. L'existence de l'équilibre de Nash

consiste à prouver que le jeu possède au moins un point d'équilibre. Nous avons déjà identifié que notre jeu est un jeu fini dont chaque joueur est associé à un ensemble fini de stratégie. En se référant à la définition de Clempner & Poznyak (2020), qui dit que chaque jeu fini possède au moins un point d'équilibre, nous pouvons dire que notre jeux en possède au moins un. Ce qui implique l'existence de l'équilibre de Nash.

2.5.2.2 Formulation pour la recherche de l'EN

Reprenant l'équation (2.30), notons par :

$$\{\mathcal{X}\} = \{\{\mathcal{X}_1\}, ..., \{\mathcal{X}_m\}, ..., \{\mathcal{X}_M\}\} \quad \forall m = 1, ..., M \tag{2.33}$$

l'ensemble des stratégies des joueurs participant au jeu.

Pour le point d'équilibre, notons par $\{\mathcal{X}^*\}$, l'ensemble des stratégies des joueurs.
Plus précisément :

$$\{\mathcal{X}^*\} = \{\{\mathcal{X}_1^*\}, ..., \{\mathcal{X}_m^*\}, ..., \{\mathcal{X}_M^*\}\} \quad \forall m = 1, ..., M \tag{2.34}$$

$$= \{\mathcal{X}_1^{*u}, ..., \mathcal{X}_m^{*v}, ..., \mathcal{X}_M^{*w}\} \tag{2.35}$$

$\{\mathcal{X}^*\}$ est alors composé par le u ème stratégie du premier joueur, .., v ème stratégie du joueur m (selon l'équation (2.30) $v \in [1, \Lambda]$), .., et la w ème stratégie du joueur M

Chaque stratégie qui définie le point d'équilibre est également associée à un coût de ressources dont son calcul dépend de la stratégie du joueur lui même et les stratégies des autres joueurs.

Nous le notons par $RC_m(\mathcal{X}_m^*, \{\mathcal{X}^{-m}\})$ où, $\{\mathcal{X}^{-m}\}$ indique l'ensemble des stratégies des autres joueurs lorsque le joueur m adopte la stratégie $\mathcal{X}_m^*$ (= $\mathcal{X}_m^{*v}$ selon l'équation 2.35)

Enfin, selon l'équation (2.23), l'ensemble des stratégies

$$\{\mathcal{X}^*\} = \{\{\mathcal{X}_1^*\}, ..., \{\mathcal{X}_m^*\}, ..., \{\mathcal{X}_M^*\}\}$$

est en équilibre de Nash si et seulement si :

$$\Leftrightarrow \qquad RC_m\left(\mathcal{X}_m^*, \{\mathcal{X}^{-m}\}\right) \geqslant RC_m\left(\mathcal{X}_m^\lambda, \{\mathcal{X}^{-m}\}\right)$$

$$\forall \mathcal{X}_m^* \in \{\mathcal{X}^*\} \bigcap \{\mathcal{X}_m\}, \quad \mathcal{X}_m^\lambda \in \{\mathcal{X}_m\}, \quad \forall m = 1...M \quad (2.36)$$

2.5.3 Matrice du jeu

Comme la formulation définie dans l'équation (2.3) est un problème d'optimisation linéaire et que tous les AP_m partagent les mêmes contraintes, et selon Stein & Sudermann-Merx (2018), ce problème est considéré comme un problème linéaire d'équilibre de Nash généralisé (LGNEP).

Le problème d'optimisation défini dans l'équation (2.24) sera généralisé en utilisant les résultats obtenus à partir de l'équation (2.31). Notons alors C_m le coût total qui comprend le coût des ressources Rc_m et le coût des liens Lc_m.

$$C_m(SP_\lambda) = Rc_m(SP_\lambda) + Lc_m(SP_\lambda) \quad (2.37)$$

Nous pouvons transformer l'équation (2.24) comme suivant :

$$\underset{\mathcal{X}_m}{Min} \sum_{\{SP_\lambda\}} RC_m\Big(\mathcal{X}_m, C_m(SP_\lambda)\Big) \quad (2.38)$$

où $\mathcal{X}_m$ est le vecteur défini dans l'équation (2.31) et que :

$$C_m(SP_\lambda) = \Big(\underbrace{C_m(SP_1), .., C_m(SP_1)}_{U_m \text{ fois}}, .., \underbrace{C_m(SP_\lambda), .., C_m(SP_\lambda)}_{U_m \text{ fois}}, ..., \underbrace{C_m(SP_\Lambda), .., C_m(SP_\Lambda)}_{U_m \text{ fois}}\Big) \quad (2.39)$$

Les contraintes liées au modèle seront aussi généralisées comme suit :

$$\left[\mathcal{A}\right] * \mathcal{X}_m + \sum_{m \neq l} \left[\mathcal{A}\right] * \mathcal{X}_l \leqslant \left[b\right] \quad (2.40)$$

Où $\mathcal{X}_l$ regroupe les décisions des autres joueurs tandis que $\mathcal{X}_m$ la décision du joueur AP_m. $\left[b\right]$ est le vecteur à droite de l'inégalité dans les équations des contraintes (2.25) à (2.29). La matrice $\left[\mathcal{A}\right]$ est la matrice formée par les coefficients des contraintes.

$$\mathcal{A} = \begin{pmatrix}
[a_1^1] & [0] & [0] & \cdots & [0] & \cdots & [0] \\
[0] & [a_2^1] & [0] & \cdots & [0] & \cdots & [0] \\
\vdots & \vdots & \vdots & & \vdots & & \vdots \\
[0] & \cdots & [a_\Lambda^1] & \cdots & [0] & \cdots & [0] \\
\vdots & \vdots & \vdots & [a_\lambda^m] & \vdots & & \vdots \\
[0] & [0] & [0] & \cdots & [a_1^M] & \cdots & [0] \\
\vdots & \vdots & \vdots & & \vdots & \ddots & \vdots \\
[0] & \cdots & [0] & \cdots & [0] & \cdots & [a_\Lambda^M] \\
-- & -- & -- & -- & -- & -- & -- \\
[r_1^1] & [0] & [0] & \cdots & [r_1^M] & \cdots & [0] \\
\vdots & \vdots & \vdots & [r_\lambda^m] & \vdots & & \vdots \\
[0] & [0] & [r_\Lambda^1] & \cdots & 0 & \cdots & [r_\Lambda^M] \\
-- & -- & -- & -- & -- & -- & -- \\
[s_1^1] & [0] & [0] & \cdots & [s_1^M] & \cdots & [0] \\
\vdots & \vdots & \vdots & [s_\lambda^m] & \vdots & & \vdots \\
[0] & [0] & [s_\Lambda^1] & \cdots & 0 & \cdots & [s_\Lambda^M] \\
-- & -- & -- & -- & -- & -- & -- \\
[t_1^1] & [0] & [0] & \cdots & [t_1^M] & \cdots & [0] \\
\vdots & \vdots & \vdots & [t_\lambda^m] & \vdots & & \vdots \\
[0] & [0] & [t_\Lambda^1] & \cdots & 0 & \cdots & [t_\Lambda^M] \\
-- & -- & -- & -- & -- & -- & -- \\
[w_1^1] & [0] & [0] & \cdots & [w_1^M] & \cdots & [0] \\
\vdots & \vdots & \vdots & [w_\lambda^m] & \vdots & & \vdots \\
[0] & [0] & [w_\Lambda^1] & \cdots & 0 & \cdots & [w_\Lambda^M] \\
-- & -- & -- & -- & -- & -- & -- \\
[1] & [1] & [1] & \cdots & [0] & \cdots & [0] \\
\vdots & \vdots & \vdots & [1] & \vdots & & \vdots \\
0 & 0 & 0 & \cdots & [1] & \cdots & [1]
\end{pmatrix} \qquad b = \begin{pmatrix}
[\tau_1] \\
[\tau_1] \\
\vdots \\
[\tau_1] \\
\vdots \\
[\tau_M] \\
\vdots \\
[\tau_M] \\
-- \\
Me^1 \\
\vdots \\
Me^\Lambda \\
-- \\
Co^1 \\
\vdots \\
Co^\Lambda \\
-- \\
St^1 \\
\vdots \\
St^\Lambda \\
-- \\
BW^1 \\
\vdots \\
BW^\Lambda \\
-- \\
U_1 \\
\vdots \\
U_M
\end{pmatrix} \tag{2.41}$$

avec :

$$[a_{\lambda}^{m}] = \begin{pmatrix} \mathcal{L}(EU_{1}^{m}|SP_{1}) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathcal{L}(EU_{U_m}^{m}|SP_{\Lambda}) \end{pmatrix} \tag{2.42}$$

$[a_{\lambda}^{m}]$ est formée par l'ensemble de la latence entre les utilisateurs de l'application m et les centres de données. Cette matrice a une dimension $k * k$ où $k = \Lambda * U_m$ (Λ est la taille de l'ensemble SP_n regroupant les centre de données des fournisseurs de services, et U_m nombre d'utilisateurs pour AP_m)

$$[r_{\lambda}^{m}] = [M_m, \quad \cdots \quad , M_m] \tag{2.43}$$

$$[s_{\lambda}^{m}] = [C_m, \quad \cdots \quad , C_m] \tag{2.44}$$

$$[t_{\lambda}^{m}] = [S_m, \quad \cdots \quad , S_m] \tag{2.45}$$

$$[w_{\lambda}^{m}] = [bw_m, \quad \cdots \quad , bw_m] \tag{2.46}$$

$r_{\lambda}^{m}, s_{\lambda}^{m}, t_{\lambda}^{m}, w_{\lambda}^{m}$ sont des vecteurs composés par les contraintes de l'application m en fonction, respectivement, de la mémoire, du CPU, du stockage et de la bande passante. Ces vecteurs ont une dimension $k = \Lambda * U_m$.

La solution générée par l'équation 2.40 nous aide à définir les stratégies des joueurs. Nous allons utiliser ces stratégies dans la partie suivante pour la recherche du point d'équilibre.

2.6 Méthode de recherche de l'équilibre de Nash

Plusieurs méthodes sont proposées par des auteurs pour la recherche de l'équilibre dans un jeu (Stein & Sudermann-Merx, 2018; Salehisadaghiani & Pavel, 2016; Melnik, 2015; Tahmasebi & Pourghassem, 2017). Certaines méthodes sont dédiées à des jeux classiques spécifiques. Nous avons pu tirer de ces références les méthodes suivantes :

- la méthode basée sur l'élimination d'une stratégie strictement dominée (Stein & Sudermann-Merx, 2018),
- la méthode basée sur la meilleure réponse du jeu (Dreves, 2019),
- la méthode de recherche utilisant l'algorithme "gossip" (Salehisadaghiani & Pavel, 2016),
- algorithme basé sur l'énumération séquentielle totalement mixte (Tahmasebi & Pourghassem, 2017),
- algorithme basé sur la recherhce du "saddle-point" (Gharesifard & Cortés, 2013).

Pour notre problème, on va s'intéresser sur les méthodes utilisées dans le cadre d'un jeu non-coopératif à N-joueurs. On va porter notre attention, particulièrement aux deux premières méthodes citées ci-dessus, à savoir, la méthode de suppression des stratégies strictement dominées (RSDS) et la méthode de la meilleur réponse (BRM).

2.6.1 Méthode de Suppression des Stratégies Entièrement Dominées (RSDS)

Cette méthode est basée sur l'élimination, étape par étape, des stratégies dominées à travers toutes les stratégies d'un joueur, étant donné qu'une stratégie dominée n'est pas la meilleure réponse pour un jeu. Le système met à jour la matrice du jeu après chaque étape d'élimination jusqu'à ce qu'il n'y ait plus de domination. La procédure d'élimination est ensuite exécutée par un processus en itération.

Considérons un joueur AP_m dans le jeu, avec l'ensemble des stratégies :

$$\{\mathcal{X}_m\} = \{\mathcal{X}_m^i, \forall i = 1, .., \Lambda\}(\Lambda \text{ nombre total de stratégie de } AP_m)$$

Chaque stratégie $\mathcal{X}_m^k$ du joueur AP_m correspond à une valeur spécifique du coût de la ressource $RC_m^k, \forall k \in \{1, .., \Lambda\}$. L'ensemble des stratégies des autres joueurs est alors désigné par : $\{\mathcal{X}^{-m}\}$

2.6.1.1 Stratégie dominée

La stratégie $\mathcal{X}_m^k$ du joueur m est dominée par la stratégie $\mathcal{X}_m^i$, considérant les stratégies des autres joueurs $\{\mathcal{X}^{-m}\}$ si le coût des ressources généré en utilisant la stratégie $\mathcal{X}_m^k$ est plus grand que le coût des ressources, en utilisant $\mathcal{X}_m^i, \forall \mathcal{X}_m^k \in \{\mathcal{X}_m\}$ et $\forall \mathcal{X}_m^i \in \{\mathcal{X}_m\}$ (Pauly, 2016).

$$RC_m^k\left(\mathcal{X}_m^k, \{\mathcal{X}^{-m}\}\right) > RC_m^k\left(\mathcal{X}_m^i, \{\mathcal{X}^{-m}\}\right) \tag{2.47}$$

2.6.1.2 Implémentation de la méthode RSDS

L'algorithme "Recursive-Backtracking" utilisé par Stein & Sudermann-Merx (2018) décrit la méthode utilisée pour rechercher le point d'équilibre d'un jeu non coopératif à N-joueurs. Cet algorithme illustre la mise en œuvre de la méthode RSDS. Le principe de cette méthode est l'élimination de toute stratégie strictement dominée parmi l'ensemble des stratégies du joueur. L'algorithme 2.1, permet d'identifier s'il y a domination entre deux stratégies. Il est ensuite utilisé par l'algorithme 2.2, pour la suppression de l'ensemble des stratégies dominées.

2.6.2 Méthode de la Meilleure Réponse (BRM)

Cette méthode est basée sur la recherche de la meilleure situation d'un joueur dans la matrice des coûts. Elle est généralement utilisée lorsque le jeu n'a pas de stratégie dominée ou que l'itération n'a pas de stratégie dominée à supprimer. La méthode BRM permet alors d'identifier l'ensemble des équilibres de Nash, qui sont les intersections de toutes les meilleures réponses de chaque joueur à travers leurs stratégies. En revanche, cette méthode trouvera toujours tous les équilibres de Nash, même si les joueurs ne possèdent pas de stratégie dominante ou dominée.

Algorithme 2.1 Identification de stratégie dominée(DSI)

Algorithme : Identification de stratégie dominée(DSI)

Function ***DSI***$\left(\{\mathcal{X}_m^1\},[RC_m^1],\{\mathcal{X}_m^2\},[RC_m^2],Domination\right)$
  Input : $\{\mathcal{X}_m^1\};[RC_m^1];\{\mathcal{X}_m^2\};[RC_m^2]$;
  Strategies and Resources Costs of a player m
  Output : $\{\mathcal{X}_m^1\}$; $[RC_m^1]$; $\{\mathcal{X}_m^2\}$; $[RC_m^2]$; Domination = false;
  var : $dom1 = 1; dom2 = 1;$
  for *All* $\{rc_{km}^1\} \in [RC_m^1]$ **do**
    if $rc_{km}^1 > rc_{km}^2$ **then**
      dom1 = dom1*0;
      Exit;
    end if
    else if $rc_{km}^2 > rc_{km}^1$ **then**
      dom2 = dom2*0;
      Exit;
    end if
  end for
  if *dom1 == 1* **then**
    $[RC_2^m] = \{\mathcal{X}_m^2\} = NULL$;
    Domination = true;
  end if
  if *dom2 == 1* **then**
    $[RC_1^m] = \{\mathcal{X}_m^1\} = NULL$;
    Domination = true;
  end if
  Return :($\{\mathcal{X}_m^1\}$,$[RC_m^1]$,$\{\mathcal{X}_m^2\}$,$[RC_m^2]$,Domination);
end

2.6.2.1 Une stratégie comme meilleure réponse

Un coût de ressources RC_m^i d'une stratégie de joueur $\mathcal{X}_m^i \in \{\mathcal{X}_m\}$ est considéré comme une meilleure réponse si ce RC_m^i a la meilleure valeur comparée aux autres selon les exigences du système et en considérant toutes les stratégies des autres joueurs $\{\mathcal{X}^{-m}\}$.

Algorithme 2.2 "Recursive Backtracking Algorithm"

Algorithme : "Recursive Backtracking Algorithm"
Adapté de Stein & Sudermann-Merx (2018)

Procedure ***Recursive-Backtracking***$(\{\mathcal{X}\}, \{RC\})$
  Input : $\{\mathcal{X}\}; \{RC\}$: Resources Cost and Strategy
  Output : $\{\mathcal{X}\}; \{RC\}$;
  Variable :
   DominationExist = false;
  for $m = 1$ *to* M **do**
    L = size of $[\mathcal{X}_m]$ of AP_m
    for l=*1 to L-1* **do**
      for k=l+*1 to* L **do**
        DSI$\left(\{\mathcal{X}_l^m\},[RC_l^m],\{\mathcal{X}_k^m\},[RC_k^m], Domination\right)$
      end for
    end for
    if *Domination* **then**
      DominationExist = true;
      Update($\{\mathcal{X}\}$, $\{RC\}$) of other players
    end if
  end for
  Return :$\{\mathcal{X}\}$, $\{RC\}$;
  if *DominationExist* **then**
    Recursive-Backtracking$(\{\mathcal{X}\}, \{RC\})$;
  end if
end

RC_m^i est la meilleure réponse pour le joueur m $\forall i \in [1, .., K_m]$ avec K_m la dimension de $\{\mathcal{X}_m\}$

$$\Leftrightarrow \quad RC_m^i = min\{RC_m^k, \forall k = 1, .., K_m\}$$
$$\text{avec } l \in [1, .., L] \text{ où } L = \prod_{n \in \{\mathcal{M}^-\}} K_n \text{ et } \{\mathcal{M}^-\} = \{1, .., m-1, m+1, .., M\} \quad (2.48)$$

2.6.2.2 Illustration de la méthode RSDS

Afin de pouvoir implémenter cette méthode, nous allons considérer les démarches ci-dessous :

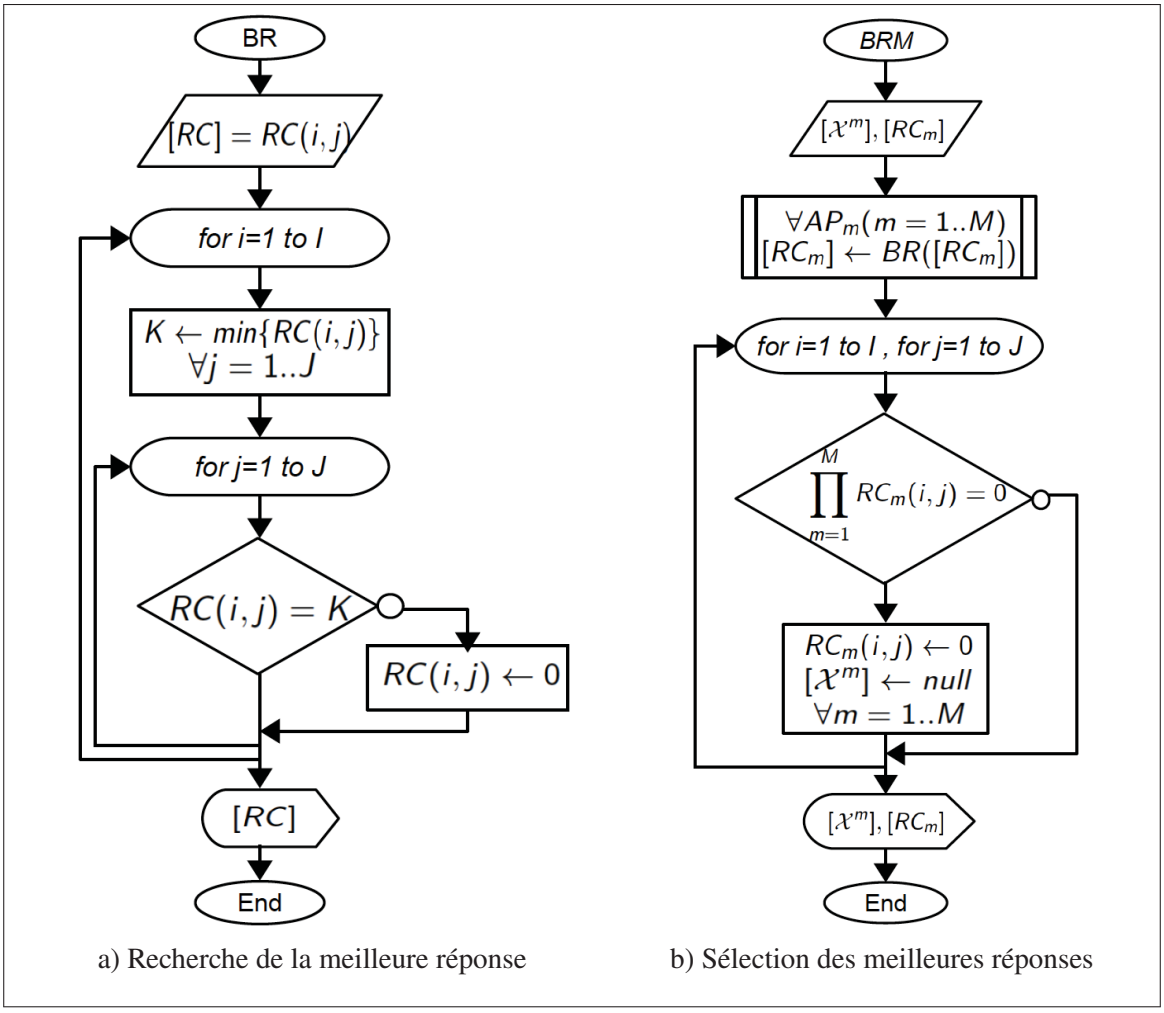
- la première étape consiste à identifier toutes les meilleures réponses dans les stratégies des joueurs,
- cette étape est itérée et appliquée à tous les joueurs,
- après que toutes les meilleures réponses soient identifiées, l'étape suivante consiste à vérifier s'il y a intersection entre les meilleures réponses de tous les joueurs. Plus clairement, si un même élément de la matrice des coûts est sélectionné comme meilleure réponse pour tous les joueurs, il sera considéré comme un point d'équilibre.
- la dernière étape consiste à identifier les stratégies correspondantes à tous les éléments sélectionnés dans la matrice des coûts,
- ces stratégies forment l'ensemble des points d'équilibre de Nash.

Les organigrammes (2.3a) et (2.3b) de la figure 2.3 illustrent l'utilisation de la méthode de sélection des meilleures réponses. L'organigramme 2.3a décrit une fonction qui identifie les meilleures réponses pour un joueur a travers sa matrice des coûts. Ceci retourne la matrice contenant seulement les éléments sélectionnés. L'organigramme 2.3b utilise cette fonction en l'itérant pour tout les joueurs. Les résultats obtenus sont ensuite superposés pour déterminer quels éléments des matrices coûts des joueurs formeront les points d'équilibre du jeu. Ces points sont ensuite retournés comme résultat final de la méthode.

2.6.3 Combinaison des deux méthodes RSDS-BRM

Les deux méthodes proposées ci-dessus nous permettent de générer les points d'équilibre d'un jeu. Leur efficacité en terme de temps de traitement dépend de la taille des matrices des coûts de ressources et stratégies des joueurs.

Ces algorithmes nous génèrent un ensemble de point d'équilibre, qui font partie de l'équilibre de Nash. Ce que nous cherchons par contre c'est un résultat plus précis, qui peut être un ensemble de point dont la taille de l'ensemble est la minimale possible. L'idéal des cas est d'obtenir un point d'équilibre unique comme résultat. Cela aide les fournisseurs d'application à prendre la meilleure décision concernant l'allocation des ressources. Pour avoir ce résultat idéal, nous



a) Recherche de la meilleure réponse b) Sélection des meilleures réponses

Figure 2.3 Illustration de l'utilisation de la méthode BRM

allons superposer les deux méthodes dans un algorithme récursive. En effet, l'une des méthodes permet de réduire la taille de la matrice des coûts, et la deuxième minimise la taille de l'ensemble des points. L'analyse de la littérature et les test que nous avons menés nous a permis de conclure qu'en terme de temps d'exécution, la méthode basée sur le RSDS est plus efficace. Par contre, en terme de précision, le BRM est plus efficace, mais affiche un temps d'exécution important surtout pour une matrice de grande taille. Nous avons décidé d'utiliser en premier le RSDS pour réduire la taille de la matrice qui va être ensuite envoyée à l'entrée de la fonction utilisant la méthode BRM. L'algorithme 2.3 illustre cette combinaison.

Algorithme 2.3 Méthode Récursive utilisant RSDS et FBR

Algorithme : Méthode Récursive utilisant RSDS et FBR

```
1  Function RSDSandFBR({X}, {RC})
       Input : {X}; {RC} : Resources Cost and Strategy
       Output : {X}; {RC};
       var : Proceed ← false
2      if There is a dominated strategy then
3          Recursive-Backtracking({X}, {RC});
4          Proceed ← true;
5      end if
6      else
7          if There is best response strategies then
8              FBR({X}, {RC});
9              Proceed ← true;
10         end if
11         else
12             Proceed ← false;
13         end if
14     end if
15     if Proceed then
16         RSDSandFBR(({X}, {RC}));
17     end if
18     else
19         Return : ({X}, {RC});
20     end if
21 end
```

2.7 Conclusion

Bref dans ce chapitre, nous avons pu formuler notre modèle en l'assimilant à un jeu de transport classique. L'élaboration du modèle, basé sur l'optimisation linéaire nous a permis de faire évoluer notre système. Nous avons pris en compte les différentes contraintes pour cette formulation. La méthode basée sur la théorie des jeux est utilisée pour le résoudre, sachant que nous l'avons formulé comme un modèle muni de plusieurs fonctions objectives Des algorithmes permettant la chercher du point d'équilibre d'un jeu sont aussi proposés pour identifier la solution optimale au problème. Dans la partie suivante, nous allons implémenter ces algorithmes et tester leur efficacité dans un environnement de simulation proche du réel.

CHAPITRE 3

RÉSULTATS EXPÉRIMENTAUX

3.1 Introduction

Ce chapitre sera consacré sur la partie pratique de notre travail. Cela consiste à simuler notre système avec des données proches du réel. En effet, c'est ici que nous allons effectuer l'implémention des modèles précédemment créés. Cela nous permettra de vérifier l'efficacité de ces modèles ainsi que de valider nos hypothèses. Notre implémentation permettra également à la vérification des méthodes utilisées ainsi que leurs validation avec les résultats obtenus.

Nous allons entamer cette partie par la conversion de nos algorithmes en code de programme qui seront exécutés dans la simulation. L'implémentation est suivie par la définition du protocole de la simulation. La suite consiste à la spécification des configurations nécessaires pour faire tourner la simulation. Pour cela, la capacité des machine virtuelles seront définies en même temps que les contraintes requises par les applications. Ces configurations sont basées sur des valeurs existantes ainsi que des valeurs expérimentales. Les résultats de la simulation seront interprétés dans dernière partie, suivi par des discussions et commentaires de ces résultats.

3.2 Implémentions

Nous allons procéder à la mise en œuvre des programmes qui traduisent notre modèle et algorithmes dans cette section. Certaines informations essentielles à l'exécution de nos programmes nécessitent une collecte de données. Les données de la latence du réseau sont des informations utiles dans cette partie.

3.2.1 Plateformes utilisées

En ce qui concerne l'outil utilisé pour le codage de nos programme ainsi que la simulation, nous avons choisi GNU Octave version 5.1.0. En effet, GNU octave est un logiciel libre et à

source ouverte, fonctionnant sur n'importe quelle plateforme (Eaton, 2018). Nous l'avons choisi pour sa robustesse et son caractère multi-plateforme. Le langage utilisé est similaire au langage utilisé avec Matlab. Certe, GNU Octave est moins puissant que Matlab mais il nous offre déjà les modules nécessaires pour notre simulation. D'autres composants de cet outil sont aussi librement téléchargeables en cas de besoin. C'est un logiciel qui ne nécessite aucune licence d'utilisation. Le logiciel GNU Octave a été installé sur une machine MacBook Pro 2.7GHz, Dual-Core Intel Core i5, avec une mémoire vive de 8GB.

Concernant le collecte des données de la latence, nous allons exposer dans cette partie le principe utilisé pour faire la mesure dans réseau en temps réel. Quelque applications sont alors nécessaire. Pour notre expérimentation, nous avons utilisé l'agent de collecte "telegraf" version 1.16. Les donnés collectées sont directement stockées dans une base de données temporelle "InfluxDB" (InfluxDataInc., 2020), ainsi que dans des fichiers de collecte pour leur utilisation dans les programmes. Ces données sont directement affichés sur le tableau de bord, "Grafana" (InfluxDataInc., 2020), qui est aussi un logiciel a source ouverte, pour l'affichage en temps réel.

3.2.2 Codage des programmes

Cette partie décrite les différents programmes que nous avons mis en œuvre pour faire tourner la simulation. En effet plusieurs couches de programmes sont exécutés en séquences dans pour notre simulation. Dans un premier temps, nous utilisons une fonction qui effectue la lecture des informations initiales disponibles. Cette fonction fait la conversion de ces informations pour générer les matrices correspondantes aux fournisseurs d'applications, $\left[AP_m\right]$ aux fournisseur de services $\left[SP_n\right]$ ainsi qu'aux utilisateurs finaux $\left[EU_u\right]$.

Les informations initiales sont des données sur les centre de données comme la capacité des machines virtuelles (Mémoire, CPU, Stockage), la quantité disponible auprès d'un centre de données, le type du centre de données (Edge, Core, Publique) ainsi que le nombre des centres de données N. Les données sur les applications, comme le nombre des fournisseurs M, les

caractéristiques d'une machine virtuelle correspondante à une application (Mémoire, Core, Stockage et latence minimum),sont aussi des informations lues au début.

Les matrices $\left[AP_m\right]$, $\left[SP_n\right]$ et $\left[EU_u\right]$ sont ensuite utilisés par un autre programme qui génère les matrices $\left[\mathcal{A}\right]$ et $\left[b\right]$ décrivant notre modèle dans l'équation 2.40 (P. 32).

Ce même programme utilise ces matrices pour en générer les stratégies des joueurs placées dans des matrices $\left[\mathcal{X}_m\right]$ correspondante à chaque joueur. Une fonction qui implémente la fonction objective de notre modèle et les stratégies générées, $\left[\mathcal{X}_m\right]$, évalue les coûts des ressources et les stockent dans la matrice des coûts $\left[RC_m\right]$. C'est ainsi que les matrices $\left[\mathcal{X}_m\right]$ et $\left[RC_m\right]$ sont formées. Le reste consiste à implémenter les deux méthodes RSDS et BRM.

Pour la méthode RSDS, une fonction fait la recherche séparé des stratégies dominées dans $\left[\mathcal{X}_m\right]$. Cette fonction implémente l'équation 2.47 (P. 36). Les stratégie dominées trouvées sont éliminées, dans le programme RSDS. Ce dernier met à jours les deux matrices $\left[\mathcal{X}_m\right]$ et $\left[RC_m\right]$, et réitère cette étape avec tous les joueurs $m = 1, .., M$.

Pour la méthode BRM, une fonction fait la recherche des meilleures réponses dans la matrice $\left[RC_m\right]$. Cette fonction implémente l'équation 2.48 (P. 38). Elle est ensuite utilisé dans une autre fonction qui garde les valeurs sélectionnées en éliminant les autres. Cette même fonction met à jour les deux matrices $\left[\mathcal{X}_m\right]$ et $\left[RC_m\right]$ après que les meilleures réponses sont identifiées. Elle est aussi réitérée pour tous les joueurs $m = 1, .., M$. La fonction BRM utilise ensuite comme entrée les $\left[\mathcal{X}_m\right]$ et $\left[RC_m\right]$ obtenues pour effectuer la superposition des $\left[RC_m\right]$. Elle garde les éléments qui sont sélectionnés, pour tout les joueurs et élimine les autres. Les matrices $\left[\mathcal{X}_m\right]$ et $\left[RC_m\right]$ sont ensuite mis à jour avec ces transformations.

Ces deux méthodes sont utilisés dans une fonction récursive, qui prend comme paramètre d'entrée $\left[\mathcal{X}_m\right]$ et $\left[RC_m\right]$. La condition de sortie de la récursion est la non existence d'une stratégie dominante, ou dominée, et l'identification de toute les meilleures réponses.

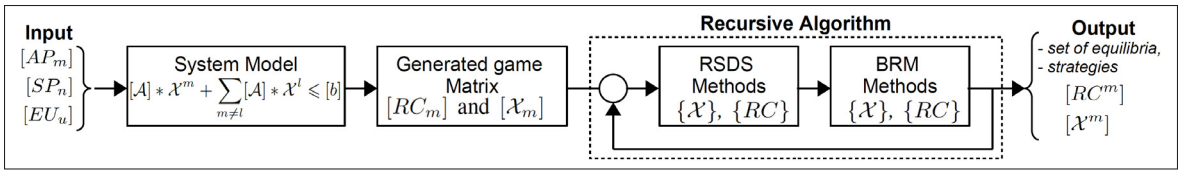


Figure 3.1 Étapes de l'implémentation du modèle

Un programme permettant de créer des fichiers CSV pour stocker les résultats ainsi d'autre programme d'itération en fonction du nombre des requêtes envoyées sont aussi utilisés. La figure 3.1, résume ces blocs de l'implémentation.

3.2.3 Protocole expérimental

Nous allons suivre les étapes suivantes pour l'exécution de nos programmes.

La première étape consiste à générer des fichiers (csv) contenant les différentes données d'entrées que les programmes vont lire. Ces fichiers contiennent les caractéristiques des fournisseurs, $\left[AP_m\right]$ et $\left[SP_n\right]$ ainsi que celles des utilisateurs finaux $\left[EU_u\right]$.

Dans la deuxième étape, nous définissons les variables qui seront utilisées par les programmes. Les valeurs de ces variables changeront au cours de l'exécution. Par exemple le nombre de requête à envoyer.

Lorsque les paramètres d'entrée sont prêts, l'étape suivante consiste à lancer le programme. Ce dernier s'exécute automatiquement et génère des fichiers au format CSV contenant les résultats de la simulation.

Les fichiers de sorties sont ensuite utilisés pour la création des graphiques représentant les résultats. Ces fichiers contiennent les points ou ensemble de points d'équilibre trouvés, correspondants aux requêtes envoyées. Ces graphiques sont représentées dans la section 3.4.

3.3 Configuration de la simulation

Les configurations des différents acteurs du système à simuler seront définis dans cette section. En effet, nous avons adopté le scénario représenté par la figure 3.2 comme exemple pour notre exécution. Dans ce scenario, notre système est constitué par 3 fournisseurs d'application définis comme suit :

- le premier fournisseur AP_1 est un fournisseur d'application "streaming" vidéo,
- le deuxième fournisseur AP_2 est un fournisseur d'application permettant d'effectuer des traitements de "big data",
- le troisième AP_3 est un fournisseur d'application pour un système de transport intelligent.

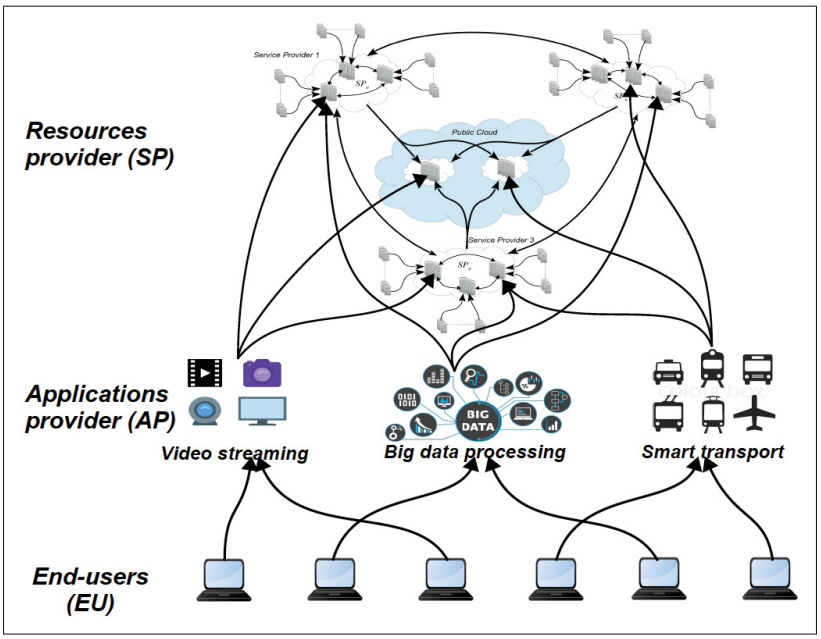


Figure 3.2 Scénario pour pour la simulation

Du coté fournisseur de services, nous avons pris en compte 3 fournisseurs de nuage privé ($\{SP_1, SP_2, SP_3\}$, ainsi que 2 fournisseurs de nuage publiques. Les trois fournisseurs privés ont chacun 3 centre de données "core". Les "core" sont chacun connectés directement à 3 centre de données "edge", (comme représentation de la figure 3.2). Le nombre des utilisateurs finaux pour chaque fournisseur d'applications caractérise les requêtes envoyés au système. Ce nombre sera considéré comme une variable que nous allons modifier pendant la simulation.

Tableau 3.1 Capacité des centres de données et exigence des applications

Fournisseur	**Type**	**CPU**	**Mem. (GB)**	**Stor. (TB)**	**BW (MB)**	**Latence (ms)**
SP_1	Core	480	480	960	1000	-
	Edge	48	48	96	100	-
SP_2	Core	640	1280	1280	1000	-
	Edge	64	128	128	100	-
SP_3	Core	560	560	1120	1000	-
	Edge	56	56	112	100	-
AP_1	Video streaming	1	1.7	0.44	10	60
AP_2	Big Data processing	2	3.75	0.48	5	40
AP_3	Smart transport	3	4	0.4	5	50

Le tableau 3.1 résume les caractéristiques des fournisseurs en terme de capacité. Dans le tableau 3.2 (P. 49) sont présentés des extraits des valeurs de la latence entre les utilisateurs de chaque application (EU_m^u) et les centres de données regroupés dans l'ensemble $\{SP_\lambda\}$. Ces valeurs sont recueillies depuis des fichiers qui seront utilisés comme des entrées des programmes.

3.4 Résultats obtenus

Cette section est consacrée à la présentation des résultats obtenus après la simulation. Les différentes graphiques présentées ici correspondent aux exécutions des algorithmes décrites dans le chapitre précédent. Nous allons diviser cette section en deux sous sections. La sous-section 3.4.2 contient les résultats de l'exécution des programmes qui génèrent les stratégies des joueurs. Dans la sous-section 3.4.3, les résultats de l'exécution des algorithmes effectuant la recherche des points d'équilibres seront présentés. L'algorithme "Recursive Backtracking Algorithm" utilisé par Stein & Sudermann-Merx (2018), qui emploi la même méthode que le RSDS, sera aussi exécuté en parallèle. Les résultats obtenus par ce dernier et notre solution seront aussi comparés et interprétés dans cette partie.

Tableau 3.2 Extrait des valeurs de la latence entre utilisateurs et centre de données

			$\mathbf{SP}_\lambda$															
		λ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	m	u	Latence entre utilisateur et centre de données $\mathcal{L}(EU_m^u \mid SP_\lambda)$ en ms															
u^{eme} utilisateur de l'application m (EU_m^u)	1	1	28	21	27	22	20	26	20	29	24	21	29	29	22	29	29	30
	1	2	28	20	24	21	26	30	22	21	28	30	29	21	22	25	23	27
	1	3	23	28	24	24	23	30	30	25	23	27	21	26	30	24	30	23
	1	4	23	21	26	29	22	24	28	24	29	29	30	29	26	26	26	30
	1	5	23	25	30	25	30	29	27	25	21	22	21	23	24	28	30	30
	1	6	21	25	21	27	24	28	20	28	23	26	26	26	29	29	27	27
	1	7	26	28	21	25	27	22	25	23	23	22	26	27	24	28	26	24
	1	8	20	23	25	28	29	24	30	23	25	24	28	30	30	21	28	22
	1	9	22	24	23	25	22	23	26	23	26	20	28	28	29	20	21	23
	1	10	25	22	30	27	27	27	26	24	22	21	30	24	26	24	23	22
	2	1	21	28	23	22	25	20	28	29	30	22	27	20	29	22	24	27
	2	2	30	24	23	25	30	29	29	24	29	20	20	24	29	24	23	29
	2	3	23	24	25	28	21	23	21	28	28	22	25	24	29	26	30	20
	2	4	27	21	23	28	23	30	30	25	21	24	27	28	23	23	22	21
	2	5	26	24	24	25	22	21	23	20	30	22	22	24	20	23	25	23
	2	6	24	25	30	30	22	30	30	25	27	25	22	29	30	29	29	29
	2	7	25	27	24	29	20	30	29	26	21	30	23	30	22	21	29	24
	2	8	29	24	30	28	27	25	23	23	21	27	23	27	26	24	29	29
	2	9	29	30	26	28	23	28	23	28	22	24	22	24	26	20	29	24
	2	10	21	21	22	22	28	25	21	22	29	25	30	28	25	27	29	25
	3	1	30	26	30	21	21	21	24	30	24	27	28	26	20	27	21	28
	3	2	25	27	26	21	23	20	23	27	27	27	26	21	28	23	24	24
	3	3	23	28	29	22	28	28	27	21	20	27	23	28	29	30	21	26
	3	4	29	20	25	28	29	24	25	26	25	30	29	26	29	29	26	23
	3	5	23	29	23	28	24	29	28	27	30	28	24	24	27	23	24	26
	3	6	25	22	29	20	27	27	23	29	23	22	25	27	22	28	20	26
	3	7	27	25	23	21	23	24	29	20	20	21	20	24	21	29	25	23
	3	8	29	22	26	29	29	27	30	28	27	24	21	20	21	20	26	27
	3	9	26	24	26	29	21	25	22	21	23	20	29	30	20	23	21	23
	3	10	29	29	20	20	28	23	25	23	22	27	23	20	25	25	23	21

3.4.1 Collecte de données de latence

Dans cette sous-section, nous allons présenter un extrait des valeurs mesurées de la latence du réseau. Ces mesures sont utiles pour identifier la latence entre un utilisateur et un serveur. Cela permet de compléter les coefficients utilisés dans les contraintes des applications. Les applications Telegraf, InfluxDB et Grafana sont utilisés pour la collecte de ces données. Telegraf

est l'agent qui effectue la collecte proprement du coté client. Les données collectée sont utilisées comme valeurs des variables d'entrées de nos programmes.

La figure 3.3 nous montre par contre l'affichage du tableau de bord affichant en temps réel les valeurs mesurées. L'exécution derrière ces résultats de collecte correspond à un exemple de mesure de latence entre un utilisateur et des centres de données des fournisseurs de service.

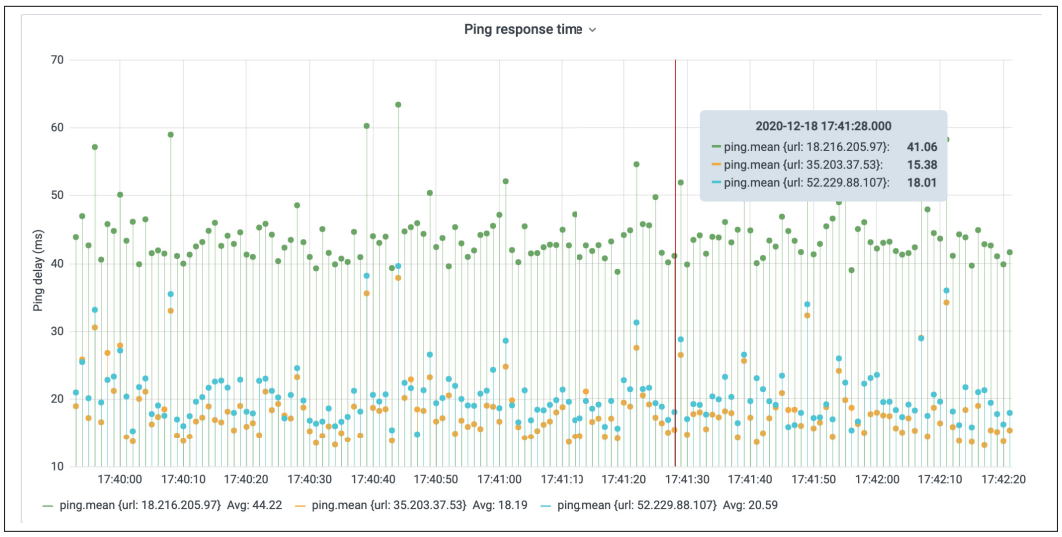


Figure 3.3 Extrait de mesure en temps réel de la latence

Le premier serveur qui a l'adresse IP : 18.216.105.97 est une machine virtuelle hébergée par "Amazon AWS". Ce serveur est localisé aux Etats-Unis, dans la ville de Colombus en Ohio. Le deuxième serveur avec l'adresse IP : 35.203.37.53 est une machine virtuelle hébergée par "Google Cloud" Le centre de donnée qui l'héberge se trouve aux Etats-Unis, dans la ville de Mountain View en Californie. Et enfin, le troisième serveur avec l'adresse IP : 52.229.88.107 est une machine virtuelle de "Microsoft Azure". Elle se trouve dans la ville de Québec au Canada.

Dans cette même figure, on peut voir les valeurs moyennes des latences mesurées, correspondantes à chaque serveur. On peut constater que ces valeurs sont influencées par le type de fournisseur, et surtout, l'emplacement géographique des serveurs. Le serveur de Google qui se trouve en Californie affiche par exemple une valeur de $18.19ms$ par contre, le serveur de Azure qui est localisé à Québec a une valeur plus grande $20.59ms$.

3.4.2 Génération des stratégies des joueurs

La génération des stratégies des joueurs consiste à former les matrices $[X_m]$ et $[RC_m]$. Elles seront ensuite utilisées dans les algorithmes RSDS et BRM comme paramètres d'entrées des différentes fonctions. Dans chaque exécution, le nombre des requêtes est varié d'une façon incrémental et aléatoire. La représentation graphique de la figure 3.4 nous montre les différents groupes de requêtes qui sont exécutées à chaque itération du programme. Dans chaque groupe le nombre d'utilisateur est correspondant au nombre des requêtes envoyé aux AP_m

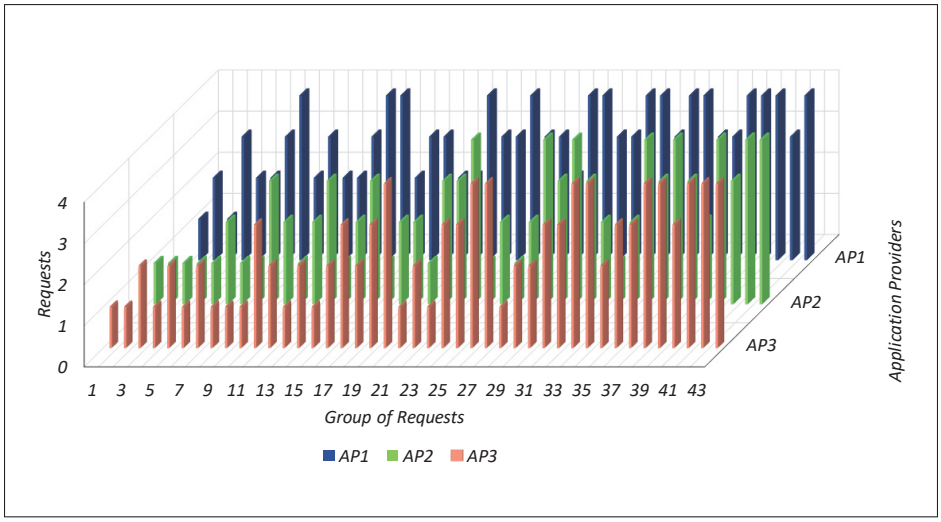


Figure 3.4 Groupe de requêtes envoyées aux fournisseurs AP_m

À chaque fois que le système reçoit une requête, le programme qui s'occupe de la génération des stratégies des AP_m est exécuté. Le temps de génération de ces stratégies est en fonction de la quantité des demandes dans la requête, qui correspond au nombre de VM demandé, ainsi que les autres paramètres comme le nombre des centres de données pouvant fournir des ressources. La figure 3.5 nous montre la courbe qui représente ce temps de génération des stratégies. On peut remarquer dans cette figure que le temps augmente de manière exponentielle avec le nombre des centres de données impliqués (n) dans la requête. Pour un petit groupe de centre de données, ($n = 3$) le temps de traitement est faible. Par contre, pour un nombre important de centre de

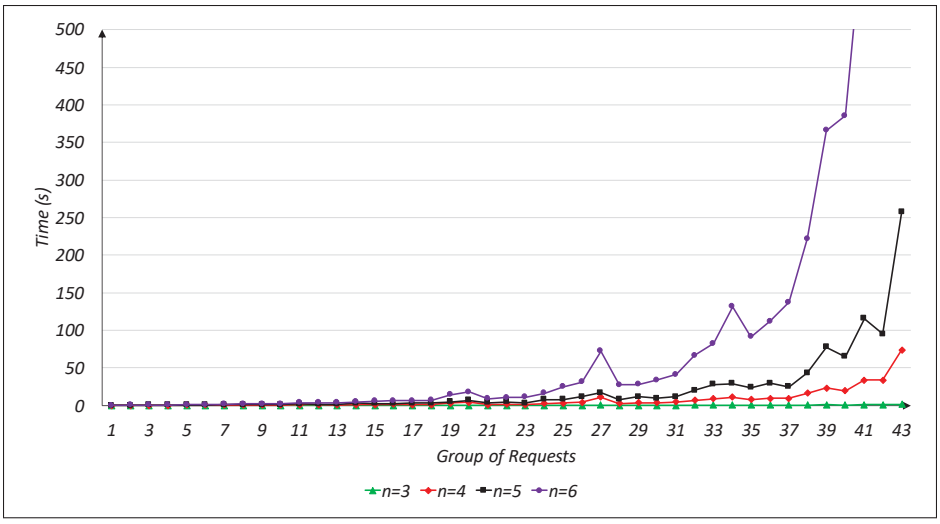


Figure 3.5 Temps de génération des stratégies des joueurs AP_m

données le temps de traitement peut grimper jusqu'à 500s. La taille de la requête influence également ce temps de génération.

Les figures 3.6 à 3.9 montrent les différentes stratégies générées. L'axe des ordonnées correspond à la taille de chaque ensemble de stratégie des joueurs (AP_m). Cette taille peut varier en fonction du groupe de requête reçu.

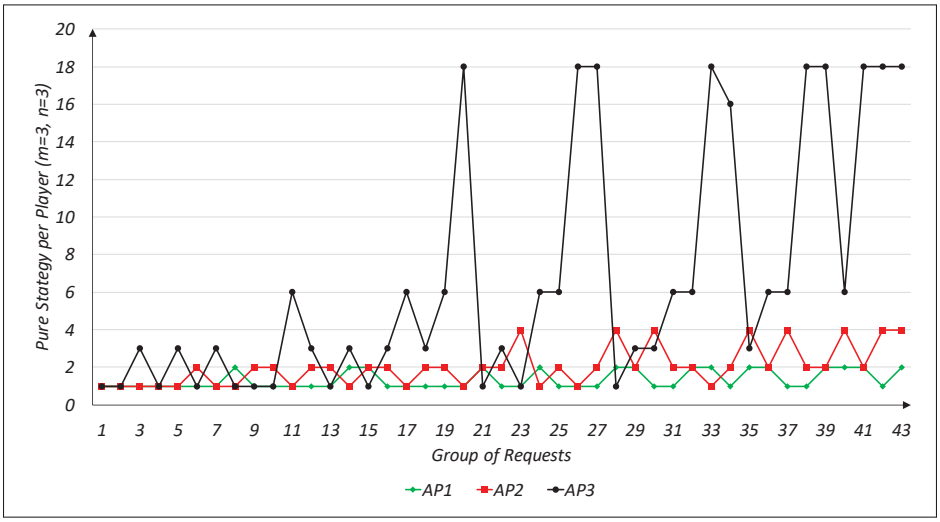


Figure 3.6 Taille des stratégies pour un ensemble de $n = 3$ DC

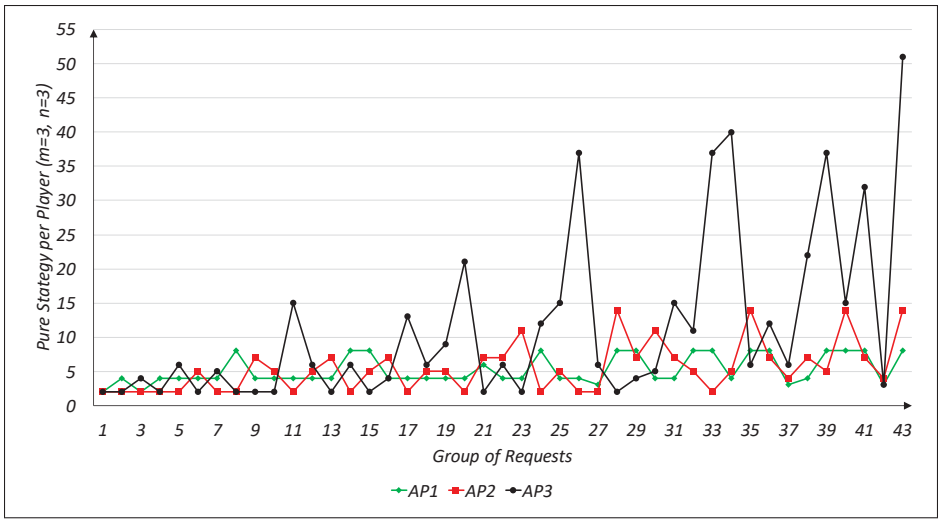


Figure 3.7 Taille des stratégies pour un ensemble de $n = 4$ DC

On peut constater dans ces graphiques que, plus le nombre des centres de données augmente, les tailles des stratégies des joueurs augmentent également. Ce phénomène illustre le modèle présenté par l'équation 2.40 (P. 32).

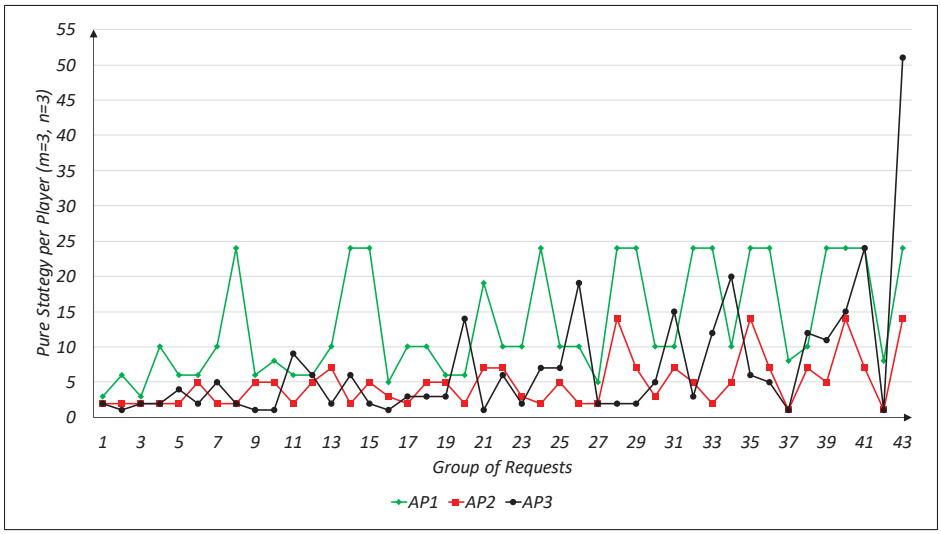


Figure 3.8 Taille des stratégies pour un ensemble de $n = 5$ DC

Dans cette équation (2.40), la dimension de la matrice du jeu $[\mathcal{A}]$ est une fonction de la taille de l'ensemble des centres de données Λ (selon la formulation 2.41). De ce fait lorsque Λ change,

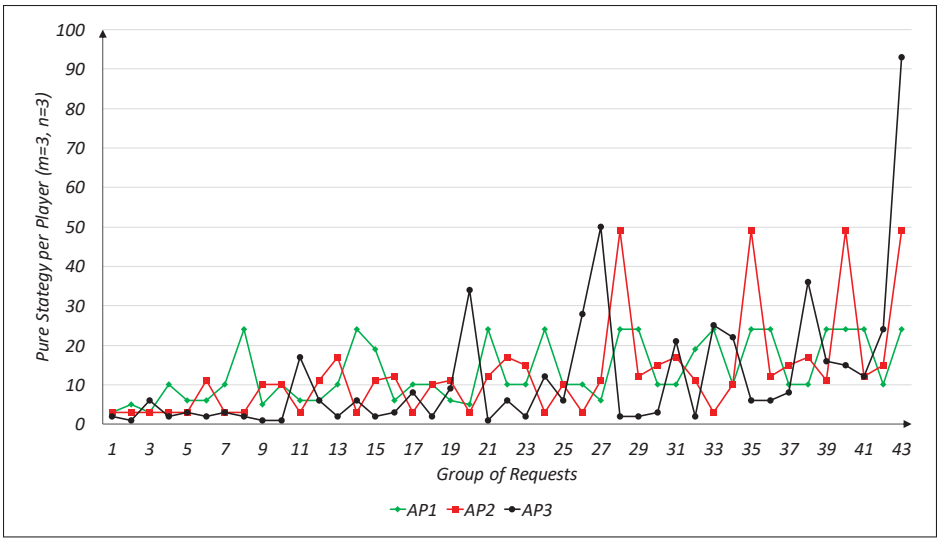


Figure 3.9 Taille des stratégies pour un ensemble de $n = 6$ DC

la dimension de la matrice change aussi. Cela a logiquement une répercussion sur le nombre des stratégies générées.

Ces résultats sont stockés sous forme de tableau, dans un fichier csv. Ils sont utilisés comme entrée des algorithme qui effectue la recherche de l'équilibre de Nash. Les sorties de ces algorithmes sont les sujets discutés dans la section suivante.

3.4.3 Recherche du point d'équilibre

Nous allons présenter les résultats de l'exécution des algorithmes de recherche du point d'équilibre du jeu dans cette partie. Comme annoncé plus haut nous allons exécuter en même temps avec nos programmes, l'algorithme utilisé par Stein & Sudermann-Merx (2018). Les deux programmes utilisent les mêmes configurations et fonctionnent dans la même condition, afin de pouvoir comparer leurs résultats. En effet les matrices coûts et stratégies générées dans la sous section 3.4.2 seront utilisées et les programmes sont également lancés sur la même plateforme.

Trois principaux éléments sont à comparer à l'issue de l'exécution de ces programmes à savoir,

- le temps d'exécution des algorithmes,

- la taille de l'ensemble des points d'équilibres obtenus,
- le nombre des stratégies des joueurs, correspondants aux des points d'équilibres.

La figure 3.10 représente le temps d'exécution des deux algorithmes. Dans cette figure, nous pouvons constater que notre algorithme affiche un temps d'exécution faiblement supérieur comparé a l'algorithme de Stein & Sudermann-Merx (2018). La raison est que nous utilisons les deux méthodes en cascade, dont la première utilise la même méthode de recherche que l'algorithme cité ci-dessus. L'écart est tout de même très petit et peut être acceptable.

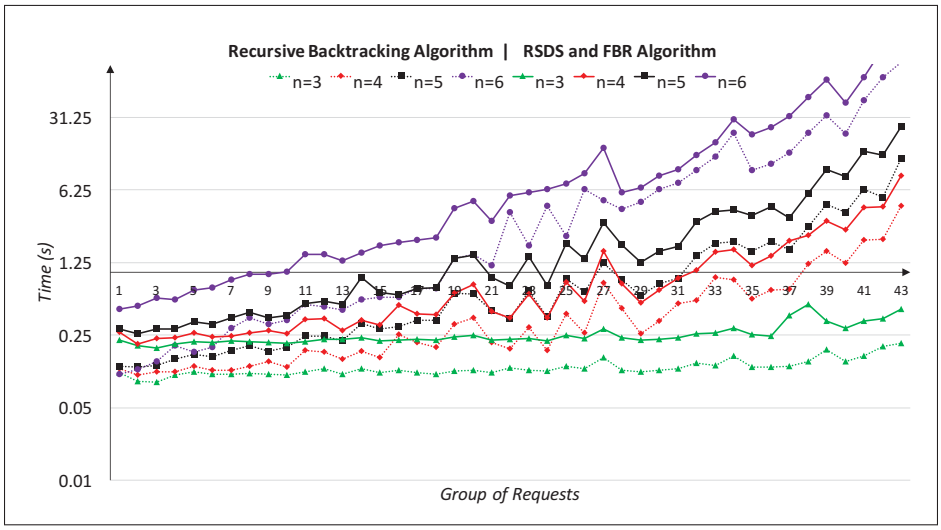


Figure 3.10 Taille des stratégies pour un ensemble de $n = 6$ DC

Comme nous l'avons déjà parlé dans le précédent chapitre, l'algorithme basé sur la méthode RSDS est avantageux en terme de temps d'exécution. Par contre, celui qui est basé sur la méthode BRM filtre très bien les points d'équilibres. En effet ce dernier peut évaluer l'équilibre en un temps record en fonction de la dimension de la matrice du jeu utilisée à l'entrée. Dans la figure 3.10, on peut remarquer que les temps d'exécutions des deux algorithmes prennent une allure linéaire. Ce qui veut dire que ces temps d'exécutions augmentent d'une façon exponentielle avec le nombre la requête, étant donné que nous utilisons une échelle logarithmique dans cette figure. Évidemment, le nombre des centres de données est une des sources de variation de ce temps.

Les figures 3.11 à 3.14 nous montrent les résultats de nos exécutions. En effet les états d'équilibre de notre jeu, qui sont, soit des points soit des ensembles de points sont présentés dans ces figures.

Ces valeurs sont aussi obtenues en variant le nombre des demandes dans une requête ainsi que le nombre centre de données impliqués.

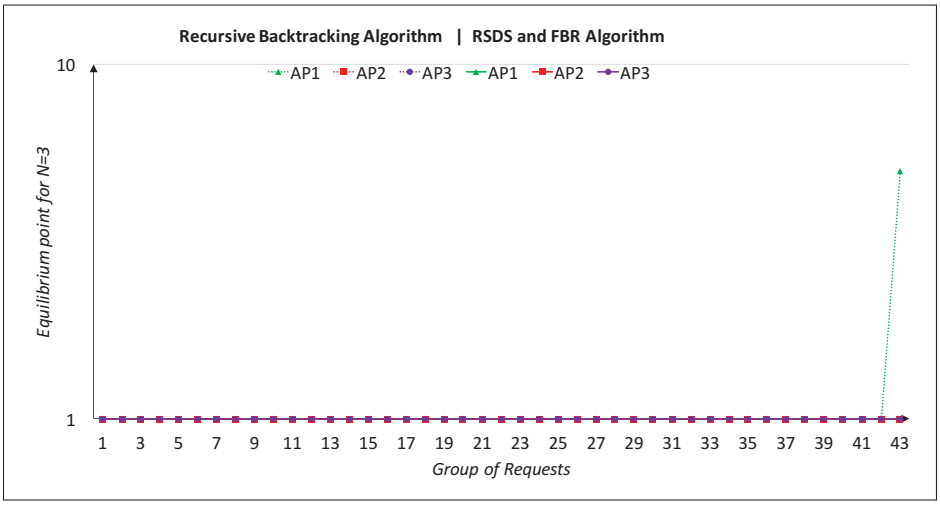


Figure 3.11 Nombre des points d'équilibre pour $N = 3$

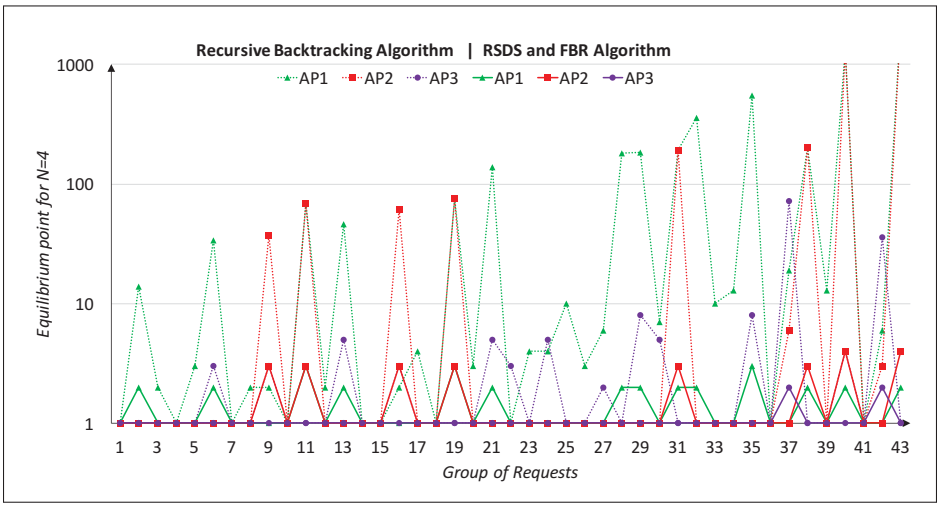


Figure 3.12 Nombre des points d'équilibre pour $N = 4$

Ces résultats démontrent l'avantage que notre solution offre. Dans la plupart des cas, notre solution évalue un point d'équilibre unique au lieu d'un ensemble de point. Cela est nécessaire

pour les fournisseurs, du fait que, ces derniers n'auront plus d'autre action à entreprendre pour la décision d'allocation. En effet, dans le cas où le résultat est formé par un ensemble de point, les fournisseurs doivent encore choisir un point associé à une stratégie dans cet ensemble. Ce qui n'est plus nécessaire dans notre cas, ou tout au moins, la décision implique quelque points seulement. Cela les orientent directement vers une décision unique sur l'endroit (centre de données) où les ressources correspondantes aux requêtes des utilisateurs seront allouées. Pour les cas où notre programme génère plus d'un point, la taille de l'ensemble des points générés ne dépasse pas 4.

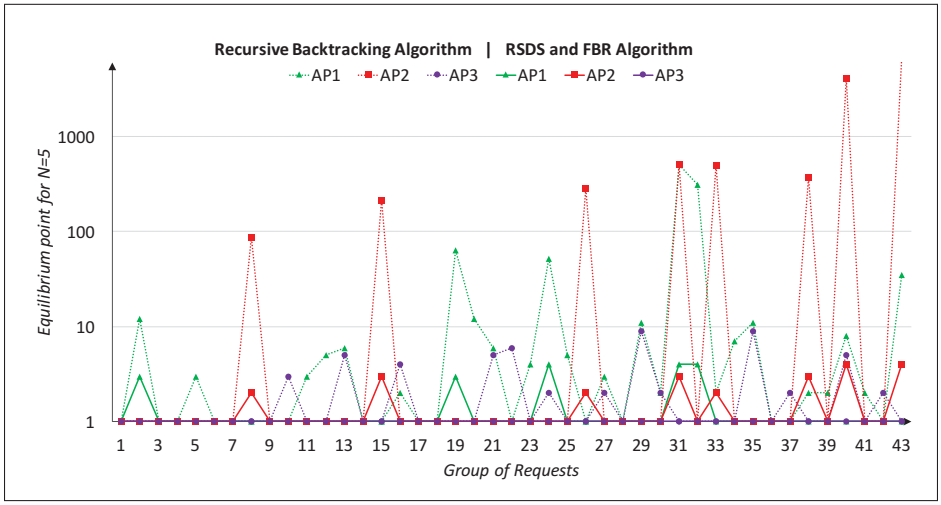


Figure 3.13 Nombre des points d'équilibre pour $N = 5$

Cette unicité de solution dépend aussi des caractéristiques de l'application offerte par les fournisseurs. Si nous regardons bien ces figures (3.11 à 3.14), sur chaque point d'équilibre, l'application AP_3 a une unique valeur, à presque 98% des cas. Seulement dans le cas où $n = 4$ (Fig. 3.12) que nous pouvons voir 2 valeurs, qui affiche chacune, deux points d'équilibres.

Bref, dans l'ensemble et pour tout les AP_m, à 90% des cas, notre solution offre un point unique d'équilibre. Cette unicité de solution apparaît dans moins de 25% des cas avec l'autre algorithme. Ce qui implique l'efficacité de l'utilisation de notre solution par rapport à ce dernier. Ces pourcentages diminuent lorsque le nombre des centres de données augmente.

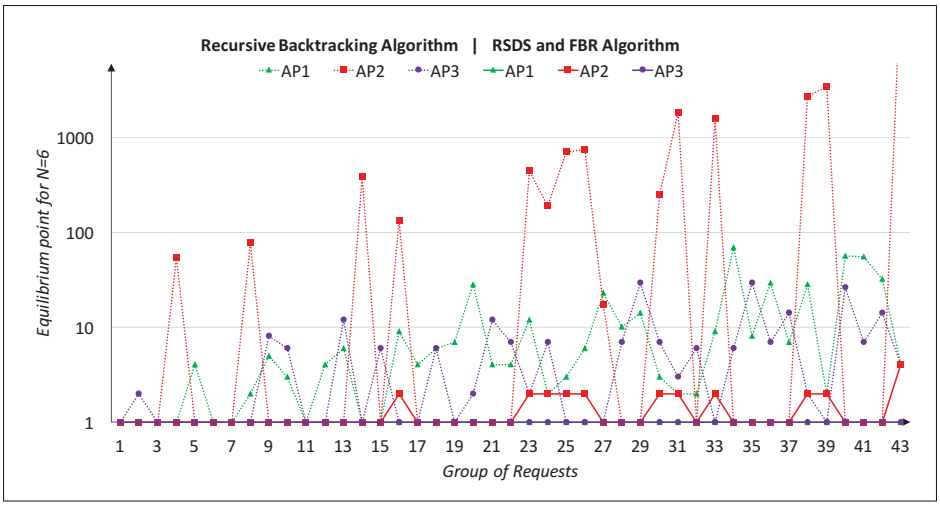


Figure 3.14 Nombre des points d'équilibre pour $N = 6$

Ces points d'équilibre sont obtenus par l'utilisation d'une ou des stratégies de chaque joueur. Les figure 3.15 à 3.18 nous montre le nombre des stratégies qui génèrent les différents points d'équilibre associés aux joueurs. Ces résultats sont en correspondances avec les points d'équilibres des précédentes figures (Fig. 3.11 à 3.14).

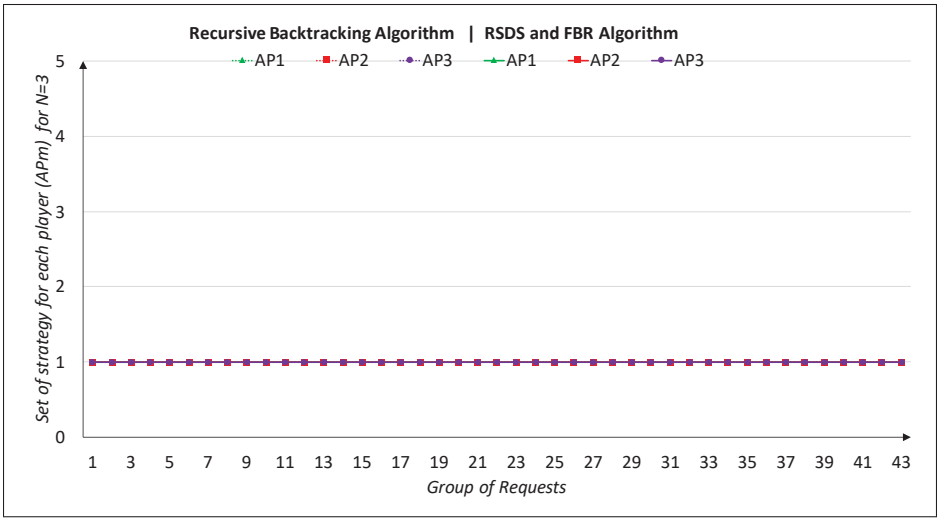


Figure 3.15 Stratégies des joueurs qui génèrent les points d'équilibre ($N = 3$)

On peut remarquer dans ces ensembles de stratégies que si l'algorithme évalue plusieurs point d'équilibre, ce même résultat peut être associé à une unique stratégie. L'explication est que si

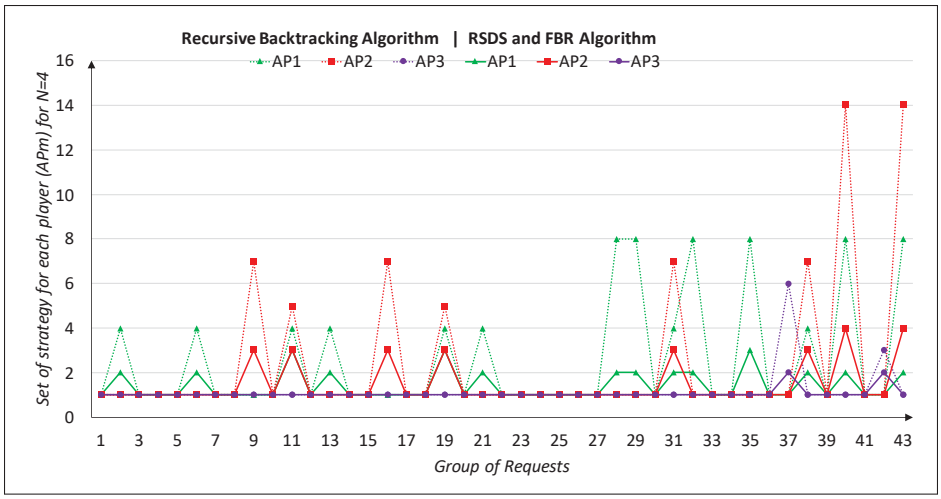


Figure 3.16 Stratégies des joueurs qui génèrent les points d'équilibre ($N = 4$)

une stratégie d'un joueur est sélectionnée dans la solution, les différents points composants la solution indique le croisement avec plusieurs stratégies des autres joueurs. Comme dans le précédent résultat, on peut voir clairement que notre solution offre dans la plupart des cas une unique stratégie pour chaque point d'équilibre.

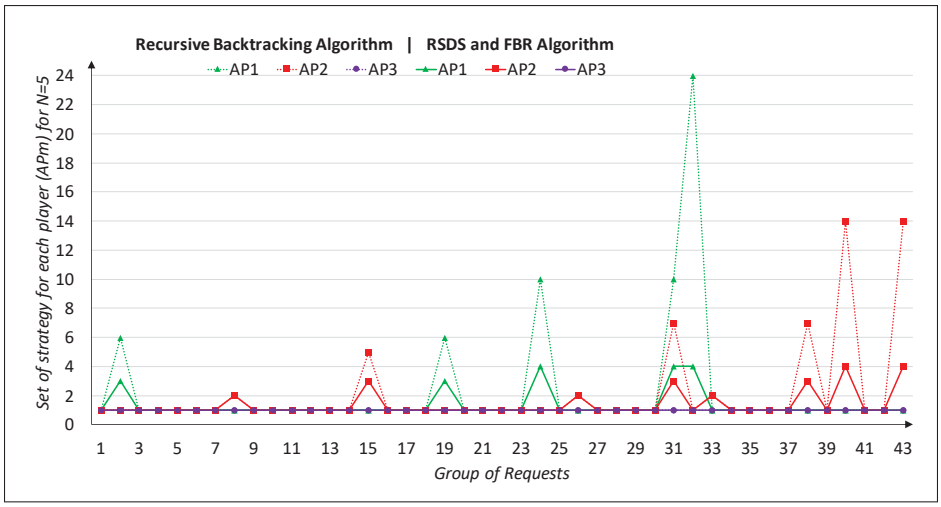


Figure 3.17 Stratégies des joueurs qui génèrent les points d'équilibre ($N = 5$)

Les 92% des résultats dans ces figures (Fig. 3.15 à 3.18) affichent que notre algorithme sélectionne une unique stratégie pour chaque joueur. Ce qui laisse seulement à 8% des cas où les fournisseurs

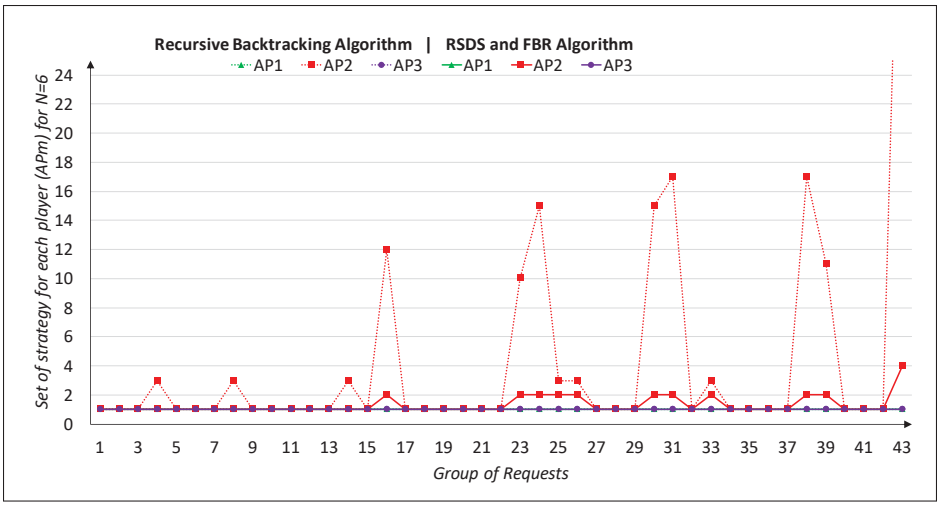


Figure 3.18 Stratégies des joueurs qui génèrent les points d'équilibre ($N = 6$)

choisissent une parmi les 2 ou 3 (au maximum 4) stratégies. Cela montre encore l'efficacité de notre solution qui confirme ce qui est déjà évoqué dans la précédente sous section.

3.5 Conclusion

Bref, dans ce chapitre, nous avons pu convertir notre modèle ainsi que les méthodologies proposées en programmes réels. On a vu aussi dans ce chapitre les extraits des valeurs des latences utilisées dans notre simulation et comment les mesurer. Une comparaison de la performance de notre algorithme avec l'algorithme déjà utilisé dans un article scientifique (Stein & Sudermann-Merx, 2018) nous a permis de connaître la faiblesse et l'avantage de notre solution.

On a remarqué que notre algorithme a sa faiblesse, sur le temps d'exécution, surtout lorsque la taille de la requête est grande. Cela nous permettra dans l'avenir de penser à une méthode nous permettant d'améliorer cette durée d'exécution. On a vu aussi dans nos résultats que dans les 92% des cas, notre solution offre une unique choix de stratégie, une valeur déjà proche de l'objectif, qui est d'en avoir le 100%.

CONCLUSION ET RECOMMANDATIONS

En guise de conclusion, les travaux que nous avons entrepris dans ce mémoire sont surtout basés sur l'utilisation des plateformes infonuagiques. Les méthodologies que nous avons utilisées nous permet d'apporter des améliorations sur les services infonuagiques, plus précisément, sur l'allocation de ressources à travers un architecture de nuage hétérogène. Les concepts utilisés dans ce document se focalisent sur la conception des modèles mathématiques pour représenter les services et mécanismes de gestion des ressources sur le nuage.

La première problématique que nous avons évoqué traite l'optimisation du coût des ressources sur des nuages hétérogènes interconnectés. Dans cela, un modèle d'interopérabilité a été conçu. Ce modèle prend en compte tous les paramètres nécessaires permettant d'obtenir un meilleur choix d'allocation. Ces paramètres dépendent des critères imposés par l'application de l'utilisateur ainsi que les configurations disponibles au sein des fournisseurs des ressources. Le modèle transforme ces critères en contraintes pour le problème d'optimisation. Nous avons utilisé la méthodologie basée sur la programmation linéaire pour sa formulation. On a aussi pris en considération dans la formulation de notre modèle la multiplicité des fournisseurs d'application. Chaque fournisseur est associé à une fonction objective qui partagent les mêmes contraintes.

Nous avons utilisé la méthodologie basées sur la théorie des jeux pour résoudre ce problème, qui est adaptée au modèle possédant plusieurs fonctions objectives. Une similarité entre notre problème et un jeu de transport classique a été constaté. De ce fait, nous avons traité notre problème de la même façon que ce type de jeu classique. Nous l'avons considéré comme un jeu non-coopératif à plusieurs joueurs. Le modèle du jeu a été défini avec ces descriptions, dont les joueurs sont composés par les fournisseurs d'application. Leurs stratégies définissent la façon dont ils allouent les ressources virtuelles, associées à des coûts. Ce que nous avons souhaité obtenir de cette formulation est la solution optimale qui tient en compte les stratégies des joueurs. Cela implique alors l'adoption d'une solution qui recherche ces stratégies optimales des joueurs.

On a alors proposé des solutions algorithmiques qui font la recherche de ces stratégies optimales. Deux méthodes sont utilisées pour cela, dont la méthode basée sur l'élimination des stratégies strictement dominées (RSDS) et la méthode de recherche des meilleures réponses (BRM). Nous avons utilisé ces deux méthodes conjointement pour la recherche de l'équilibre de Nash de notre jeu. L'obtention de ces points d'équilibre nous a permis d'évaluer les stratégies des joueurs qui sont les stratégies optimales du jeu.

Dans le dernier chapitre de notre travail, nous avons effectué l'implémentation et la simulation de notre modèle. Le scénario que nous avons choisi et les différentes configurations utilisées sont proches de la réalité. Nous avons fait la simulation de notre solution en parallèle avec la solution proposée par d'autre chercheur. Les résultats obtenus par les deux solutions ont été comparés. Ces résultat nous ont permis d'évaluer la performance de notre modèle. En terme d'efficacité de recherche des stratégies optimales des joueurs, notre solution évalue, sur les 92% des cas, une solution avec une unique stratégie choisie pour chaque joueur. Cette performance est déjà meilleure par rapport à la solution proposée dans notre référence. L'idéale est par contre d'en obtenir une solution plus proche de la valeur de l'efficacité à 100%. Par contre, en terme de temps d'exécution, notre solution nécessite quelque amélioration.

Bref, la suggestion d'amélioration que nous proposons dans ce travail est d'adopter une méthode de perfectionnement du temps d'exécution de l'algorithme. Il est possible d'imaginer l'envoie groupée des requêtes (avec un nombre limité de demande), étant donné que le temps d'exécution augmente exponentiellement avec la taille des requêtes. La méthodologie basée sur l'apprentissage automatique (Machine Learning) et le "Deep Reinforcement Learning" (DLR) serait aussi une bonne alternative pour cela. Cela permettra d'augmenter l'efficacité temporelle et la précision des solutions.

Un article intitulé "A Non-Cooperative transportation game to optimize resource allocation in edge-cloud environment" est associé à ce travail. Cet article a été présenté durant la conférence

"International Symposium on Networks, Computers and Communications (ISNCC 2020)" qui s’est tenue à Montréal (QC) Canada, le 20 au 22 octobre 2020. L’article est déjà publié sur IEEE Xplore (DOI: 10.1109/ISNCC49221.2020.9297302).

BIBLIOGRAPHIE

Cao, Z., Zhang, H., Liu, B. & Sheng, B. (2018). A game-theoretic framework for revenue sharing in edge-cloud computing system. *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8.

Chen, Y., Li, Z., Yang, B., Nai, K. & Li, K. (2020). A Stackelberg game approach to multiple resources allocation and pricing in mobile edge computing. *Future Generation Computer Systems*.

Clempner, J. B. & Poznyak, A. S. (2020). Finding the Strong Nash Equilibrium : Computation, Existence and Characterization for Markov Games. *Journal of Optimization Theory and Applications*, 186(3), 1029–1052.

Di Martino, B., Cretella, G. & Esposito, A. (2015). Cloud Portability and Interoperability. Dans *Cloud Portability and Interoperability* (pp. 1–14). Springer.

Dreves, A. (2019). A best-response approach for equilibrium selection in two-player generalized Nash equilibrium problems. *Optimization*, 68(12), 2269–2295.

Eaton, J. W. (2018). GNU Octave. *https ://www.gnu.org/software/octave/index*.

Gharesifard, B. & Cortés, J. (2013). Distributed convergence to Nash equilibria in two-network zero-sum games. *Automatica*, 49(6), 1683–1692.

Hu, P., Dhelim, S., Ning, H. & Qiu, T. (2017). Survey on fog computing : architecture, key technologies, applications and open issues. *Journal of network and computer applications*, 98, 27–42.

InfluxDataInc. (2020). InfluxDB Time Series Platform | InfluxData. *https ://www.influxdata.com/*.

Jonas, E. et al. (2019). Cloud programming simplified : A berkeley view on serverless computing. *arXiv preprint arXiv :1902.03383*.

Mashayekhy, L., Nejad, M. M. & Grosu, D. (2015). Cloud Federations in the Sky : Formation Game and Mechanism. *IEEE Transactions on Cloud Computing*, 3(1), 14-27. doi : 10.1109/TCC.2014.2338323.

Melnik, A. V. (2015). Equilibrium in a transportation game. *Automation and Remote Control*, 76(5), 909–918.

Pauly, A. (2016). The computational complexity of iterated elimination of dominated strategies. *Theory of Computing Systems*, 59(1), 52–75.

Qiu, C., Shen, H. & Chen, L. (2016, April). Probabilistic demand allocation for cloud service brokerage. *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pp. 1-9. doi : 10.1109/INFOCOM.2016.7524611.

Raoof, O. & Al-Raweshidy, H. (2015). Theory of Games : An Introduction. Dans *Game Theory*. InTech.

Salehisadaghiani, F. & Pavel, L. (2016). Distributed Nash equilibrium seeking : A gossip-based algorithm. *Automatica*, 72, 209–216.

Samaan, N. (2014). A Novel Economic Sharing Model in a Federation of Selfish Cloud Providers. *IEEE Transactions on Parallel and Distributed Systems*, 25(1), 12-21. doi : 10.1109/TPDS.2013.23.

Satyanarayanan, M., Bahl, P., Caceres, R. & Davies, N. (2014). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4).

Shi, W., Cao, J., Zhang, Q., Li, Y. & Xu, L. (2016). Edge computing : Vision and challenges. *IEEE Internet of Things Journal*, 3(5), 637–646.

Sliwinski, J. & Zick, Y. (2017). Learning Hedonic Games. *IJCAI*, pp. 2730–2736.

Stein, O. & Sudermann-Merx, N. (2018). The noncooperative transportation problem and linear generalized Nash games. *European Journal of Operational Research*, 266(2), 543–553.

Tahmasebi, A. & Pourghassem, H. (2017). Robust intra-class distance-based approach for multimodal biometric game theory-based rank-level fusion of ear, palmprint and signature. *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*, 41(1), 51–64.

Wu, Y., Wu, C., Li, B., Zhang, L., Li, Z. & Lau, F. C. M. (2015). Scaling Social Media Applications Into Geo-Distributed Clouds. *IEEE/ACM Transactions on Networking*, 23(3), 689-702. doi : 10.1109/TNET.2014.2308254.

Ye, M., Hu, G. & Lewis, F. L. (2018). Nash equilibrium seeking for N-coalition noncooperative games. *Automatica*, 95, 266–272.

Zafari, F., Li, J., Leung, K. K., Towsley, D. & Swami, A. (2018). A Game-Theoretic Approach to Multi-Objective Resource Sharing and Allocation in Mobile Edge Clouds.

Zaw, C. W., Tran, N. H., Saad, W., Han, Z. & Hong, C. S. (2020). Generalized Nash Equilibrium Game for Radio and Computing Resource Allocation in Co-located MEC. *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6.

Zhang, P., Zhou, M. & Wang, X. (2020). An Intelligent Optimization Method for Optimal Virtual Machine Allocation in Cloud Data Centers. *IEEE Transactions on Automation Science and Engineering*.

Zhao, J., Li, Q., Gong, Y. & Zhang, K. (2019). Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks. *IEEE Transactions on Vehicular Technology*, 68(8), 7944–7956.