République Algérienne Démocratique et Populaire
Université Abou Bakr Belkaid Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option : Réseaux et Systèmes Distribués (RSD)

Thème

Equilibrage de charge dans les systèmes distribués : application au Cloud computing

Réalisé par :
Mokhtari Ferial

Soutenu le 23 juin 2020 **devant le jury composé de :**

Mr LEHSAINI Mohamed	**Président**
Mr BENMOUNA Youcef	**Examinateur**
Mr BENMAMMAR BADR	**Encadrant**
Mr HAKEM Mourad	**Co-Encadrant**

Année universitaire 2019/2020

Remerciement

Avant tout je remercie le bon DIEU de m'avoir donné la force et la santé pour pouvoir terminer ce modeste projet de fin d'étude.

Je voudrais adresser toute ma gratitude à Mr Badr Benmammar pour avoir accepté de m'encadrer dans cette étude.
Je le remercie pour son implication, sa patience, sa disponibilité tout au long de la rédaction du mémoire.

Je souhaite également remercier Mr Mourad Hakem d'avoir su me faire confiance et m'avoir conseillée tout au long de ce PFE.

Merci à Mr Abderrazak Semmoud qui a eu l'amabilité de répondre à mes questions et de fournir les explications nécessaires.

Je souhaite également remercier Mr Mohamed Lehsaini de qui j'ai beaucoup appris,et Mr Youcef Benmouna d'avoir accepté d'évaluer mon travail au sein du jury de soutenance.

Je souhaite particulièrement remercier mon meilleur ami Yousouf Taghzouti pour son aide et son soutien moral et intellectuel tout au long de ma démarche.

Merci à ma mère et Imed Eddine, qui ont toujours su être présents.

Merci beaucoup.

Dédicaces

Je dédie ce projet a :

A ma chère mère,
Aucune dédicace ne saurait exprimer mon respect, mon amour éternel et ma considération pour les sacrifices que tu m'as consenti pour mon instruction et mon bien être.

A ma grand mère

A ma soeur Sarah et ses petits anges Wadoud, Ritedj et Ikram.

A mon meilleur ami Youc

A Imed et Imane, la navette sans vous n'aurait pas été aussi amusante.

A mes copines Amel, Ibtissam, Yasmine et Sabrine.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

AWS	**A**mazon **W**eb **S**ervices
BF	**B**rute **F**orce
CPU	**C**entral **P**rocessing **U**nit
EC2	**E**lastic **C**ompute **C**loud
EDC	**E**quilibrage **D**e **C**harge
GUI	**G**raphic **U**ser **I**nterface
IHM	**I**nterface **H**omme **M**achine
LPT	**L**argest **P**rocessing **T**ime
MSP	**M**anaged **S**ervice **P**rovider
NDL	**N**on **D**ivisible **L**oads
RAM	**R**andom **A**ccess **M**emory
VMM	**V**itual **M**achine **M**anager
VM	**V**irtual **M**achine

Résumé :

La quantité de données à stocker et à traiter dans le Cloud augmente de jour en jour et d'une façon très rapide, ce qui nécessite un mécanisme qui permet d'équilibrer la charge de stockage. Dans le cadre de ce travail, nous avons implémenté et évalué un nouvel algorithme distribué pour les charges indivisibles appliqué au Cloud Computing. L'algorithme est inspiré du problème de sacs à dos multiple. Le simulateur CloudSim Plus a été utilisé dans le cadre de ce travail et notre approche a été évaluée à l'aide de deux stratégies ; à savoir : LPT (Largest Processing Time first) et BF (Brute Force). Les résultats obtenus montrent la supériorité de LPT par rapport à BF en termes de différentes métriques et aussi pour le passage à l'échelle.
Mots clés : Système distribué, Cloud computing, Équilibrage de charge, LPT, BF.

Abstract :

The amount of data to store and process in the Cloud is increasing day by day and very quickly, which requires a mechanism to balance the storage load. As part of this work, we implemented and evaluated a new self-stabilizing algorithm for indivisible loads applied to Cloud Computing. The algorithm is inspired by the multiple knapsack problem. The CloudSim Plus simulator was used in this work and our approach was evaluated using two strategies ; namely : LPT (Largest Processing Time first) and BF (Brute Force). The results obtained show the superiority of LPT compared to BF in terms of different metrics and also for scalability.

Keywords : Distributed system, Cloud computing, Load balancing, LPT, BF.

ملخص :

يزداد حجم البيانات المخزنة والمعالجة فيزداد حجم البيانات التي يتم تخزينها ومعالجتها في السحاب يومًا بعد يوم وبسرعة كبيرة ، الأمر الذي يتطلب آلية لموازنة حمل التخزين. كجزء من هذا العمل ، قمنا بتطبيق وتقييم خوارزمية جديدة لتحقيق الاستقرار الذاتي للأحمال غير القابلة للتجزئة المطبقة على الحوسبة السحابية. الخوارزمية مستوحاة من مشكلة الحقائب المتعددة. تم استخدام محاكي CloudSim Plus في هذا العمل وتم تقييم نهجنا باستخدام استراتيجيتين ؛ وهي: LPT و BF . أظهرت النتائج التي تم الحصول عليها تفوق LPT مقارنة مع BF من حيث المقاييس المختلفة وكذلك للتوسع.
الكلمات المفتاحية : النظام الموزع ، الحوسبة السحابية ، موازنة التحميل، LPT ، BF .

Introduction générale

1. **Motivations**
   Depuis l'arrivée des ordinateurs et leurs démocratisations dans notre vie quotidienne, les besoins des utilisateurs ne cessent d'augmenter et pour y rependre les utilisateurs étaient dans l'obligation d'acheter des ressources plus puissantes et de payer le prix fort. Mais avec l'arrivée d'internet, les chercheurs ont conçus ce qu'on appelle le « Cloud computing » ce qui se résume dans l'utilisation des ressources matérielles et logicielles dans un environnement virtualisé et distribué sans avoir à le posséder physiquement. Malheureusement, une charge assez grande liée à une forte utilisation de ressources sur ce Cloud peut le rendre inefficace. Donc le nouveau défi maintenant est de créer un équilibrage entre les adhérents de cette nouvelle technologie. D'un coté les fournisseurs qui veulent maximiser le temps d'exploitation de leurs ressources afin d'avoir plus de revenu et de l'autre coté les clients qui pour leurs parts veulent minimiser leurs dépenses tout en préservant leurs exigences en termes de qualité de services.
2. **Contributions**
   Dans le cadre de ce projet de fin d'études, nous avons considéré le problème d'allocation de tâches/requêtes sporadiques, caractérisées par leurs urgences temporelles. Nous avons donc simulé un algorithme d'équilibrage de charge appliqué au Cloud computing en utilisant le simulateur CloudSim Plus. Nous avons réalisé un jeu de tests pour prouver les performances et l'efficacité de cet algorithme et nous avons exposé les résultats obtenus dans une IHM avec divers métriques de comparaison.
3. **Organisation du manuscrit**
   Notre rapport est construit autour de trois chapitres qui sont organisés comme suit :

Le premier chapitre donne un aperçu global sur le Cloud computing qui est le contexte dans lequel s'est déroulé notre PFE. Nous allons présenter la définition du Cloud, son architecture, ses types, la notion de virtualisation, les avantages du Cloud ainsi que ses inconvénients.
Par la suite nous enchaînons par le deuxième chapitre qui présente l'équilibrage de charges dans les systèmes distribués qui est la technique adoptée pour la réalisation de notre travail.
Nous allons présenter ses approches, ses politiques, ses mécanismes, quelques algorithmes standards pour sa mise en œuvre ainsi que les indices de performance utilisés dans ce domaine.
Le troisième chapitre sera consacré à la présentation de notre contribution dans le cadre de ce projet de fin d'études. Cette contribution concerne l'implémentation et l'évaluation de l'algorithme d'équilibrage de charges appliqué au Cloud computing en utilisant le simulateur CloudSim Plus.

.

Chapitre 1

Cloud computing

1.1 Introduction

Alors qu'Internet et les technologies de réseau ont progressé dans la sophistication et la fiabilité, les ingénieurs ont créé un nouveau mode de gestion des services informatiques dénommés les systèmes distribués. Ces derniers, au lieu de centraliser les données et la puissance de calcul dans un seul endroit, puis les envoyer aux clients, ils répartissent les données et les tâches de calcul sur plusieurs nœuds qui fonctionnent ensemble, en adoptant par exemple l'une de ces trois architectures les plus répandues [1] :

— **Cluster :** qui est généralement un concept de plusieurs serveurs qui fonctionnent ensemble, divisant généralement la charge entre eux pour croitre la capacité de calcul, et le rend plus disponible [2].

— **Grille :** que fait souvent référence à un ensemble de serveurs qui fonctionnent ensemble sur un calcul massif donné. Au lieu de simplement répartir la charge de travail provenant de nombreux clients, ils divisent un seul travail en sous-parties, fournissant au travail le total des ressources disponibles (ou au moins un ensemble dédié de serveurs qui est un sous-ensemble de la grille) [3].

— **Cloud :** dont son principal objectif est l'utilisation efficace des ressources. Ces dernières sont allouées à la demande, puis libérées dans le pool pour répondre à d'autres besoins. Le Cloud est virtualisé, dans

le sens ou la charge de travail n'est pas au courant de l'emplacement physique réel et des ressources qui le traitent. De plus, l'utilisateur n'est pas autorisé à accéder directement à ces ressources physiques, mais se voit plutôt attribuer un accès aux serveurs logiques qu'il consomme. Enfin, une architecture Cloud permet aux charges de travail d'évoluer en demandant plus de ressources en cas de besoin [3].

1.2 Définition

Le Cloud computing se traduit littéralement par l'informatique dans les nuages, faisant référence aux technologies d'Internet qui sont souvent représentées schématiquement par un nuage. C'est un concept abstrait qui regroupe un ensemble de technologies qui servent à délivrer des services. Son but est de pousser les entreprises à externaliser les ressources numériques qu'elles stockent, ces ressources offrent des capacités de stockage et de calcul, des logiciels de gestion de messagerie, et d'autres services sont mis à disposition par des sociétés tierces et accessibles, grâce à un système d'identification, via un PC et une connexion à Internet [4].

1.3 Architecture de Cloud computing

L'architecture de Cloud computing se base sur un ensemble de composants [5] :

— **Front end** : Utilisé par le client. Il comporte les interfaces et les applications nécessaires pour accéder aux plateformes du Cloud. Elles peuvent être des serveurs Web (y compris Chrome, Firefox, etc.), les clients légers et lourds, ou les appareils mobiles.

— **Back end** : Utilisé par le fournisseur de services. Il gère toutes les ressources nécessaires pour délivrer les services de Cloud computing.

— **Composants d'architecture de Cloud computing** :

  o **Infrastructure coté client :** L'infrastructure client est un composant frontal, qui fournit une interface graphique (GUI) pour interagir avec le Cloud.

o **Application :** L'application peut désigner n'importe quel logiciel ou plate-forme auquel un client souhaite accéder.
o **Service :** Un service Cloud qui manœuvre le type de service suivant les besoins dictés par le client.

— **L'environnement d'exécution Cloud :** Fournit l'environnement d'exécution aux machines virtuelles.

— **Stockage :** L'un des composants les plus importants du Cloud computing. Il offre une énorme quantité de capacité de stockage dans le Cloud pour stocker et gérer les données.

— **Infrastructure :** L'infrastructure Cloud comprend des composants matériel et logiciel tels que des serveurs, du stockage, des périphériques réseau, des logiciels de virtualisation et d'autres ressources de stockage nécessaires pour prendre en charge le modèle de Cloud computing.

— **Gestion :** La gestion est utilisée pour gérer les composants tels que les applications, les services, le Cloud d'exécution, le stockage, l'infrastructure et d'autres problèmes de sécurité dans le backend et établir une coordination entre eux.

— **Sécurité :** La sécurité est un composant principal intégré du Cloud computing. Il implémente un mécanisme de sécurité pour le rendre plus fiable.

— **Internet :** Internet est le moyen par lequel le front-end et le back-end peuvent interagir et communiquer entre eux.

La figure 1.1 représente l'architecture de Cloud computing.

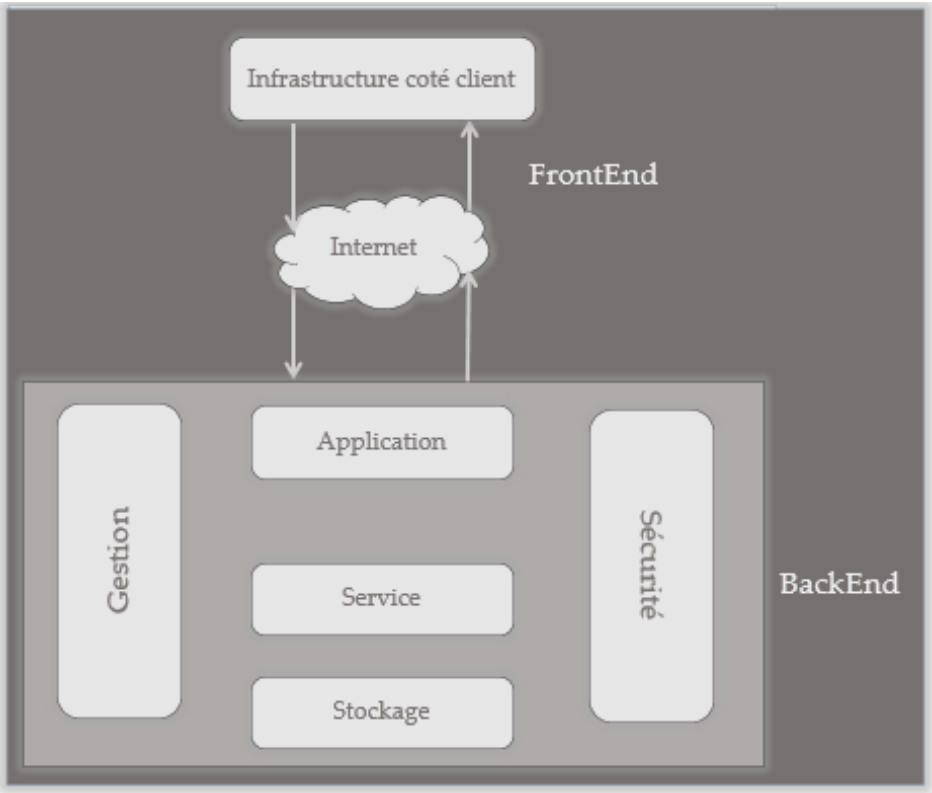

FIG. 1.1: Architecture de Cloud computing.

1.4 Types des services de Cloud computing

1.4.1 SAAS (Software As A Service)

Les utilisateurs disposent d'un logiciel prêt à l'emploi. Tous les services sont gérés par le fournisseur de service (managed services provider ou MSP), et les utilisateurs ont uniquement accès au logiciel via le site Web ou l'application mobile [6].

Exemple : Google Apps, Salesforce Dropbox, Slack, HubSpot, Cisco WebEx.

1.4.2 PAAS (Platform As A Service)

Un modèle qui donne accès à des plateformes basées sur le Cloud par exemple des systèmes d'exploitation, des systèmes de gestion de base de données, des instruments de développement et des tests de logiciel [6].

Exemple : Windows Azure, Force.com, Magento Commerce Cloud, OpenShift.

1.4.3 IAAS (Infrastructure As A Service)

Un modèle qui donne accès aux ressources informatiques responsable sur la gestion des données des applications, des middlewares et des environnements d'exécution [6].

Exemple : Amazon Web Services (AWS) EC2, Google Compute Engine (GCE), Cisco Metapod.

La figure 1.2 représente les types des services de Cloud computing.

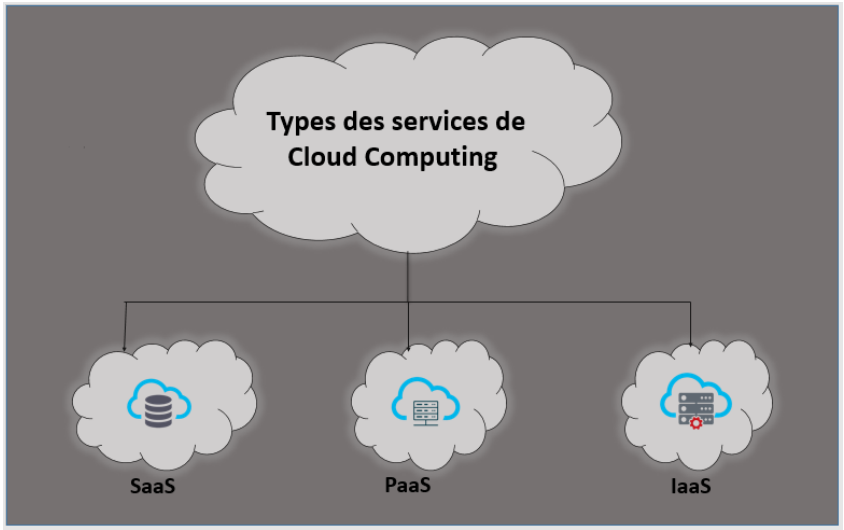

FIG. 1.2: Types des services Cloud computing.

1.5 Types de Cloud computing

1.5.1 Cloud public

Dans un Cloud public, tout est stocké et accessible via Internet. Ce système de déploiement permet à toute personne disposant des autorisations appropriées d'accéder à certaines applications et ressources. La partie la plus attractive du Cloud public est que vous ne possédez aucun des composants qui y sont présents, que ce soit le matériel, les logiciels ou les applications. Tous les composants ici sont gérés par le fournisseur. Amazon Web Services et Microsoft Azure sont deux exemples flagrants du Cloud public [7].

1.5.2 Cloud privé

Un Cloud privé est utilisé exclusivement dans les organisations qui peuvent exécuter localement ou choisir de sous-traiter à d'autres fournisseurs de services Cloud. Cette infrastructure fonctionne strictement sur un réseau privé, ce qui signifie exclusivement les personnes présentes sur le réseau peuvent y accéder, le Cloud VMware (leader mondial des infrastructures de virtualisation et de Cloud computing) et certains des produits Amazon Web Services sont quelques exemples du Cloud privé [8].

1.5.3 Cloud hybride

C'est probablement la forme fascinante du Cloud computing qui contient les fonctionnalités des Clouds publics et privés. Les organisations utilisant le Cloud hybride peuvent choisir de conserver certaines de leurs données localement et d'autres sur le Cloud. La NASA est l'exemple le plus connu d'une organisation qui utilise un Cloud hybride. Il utilise un Cloud privé pour stocker des données sensibles et utilise le Cloud public pour enregistrer et partager des données qui peuvent être consultées par le grand public dans le monde entier [9].

La figure 1.3 représente les différents types de Cloud computing.

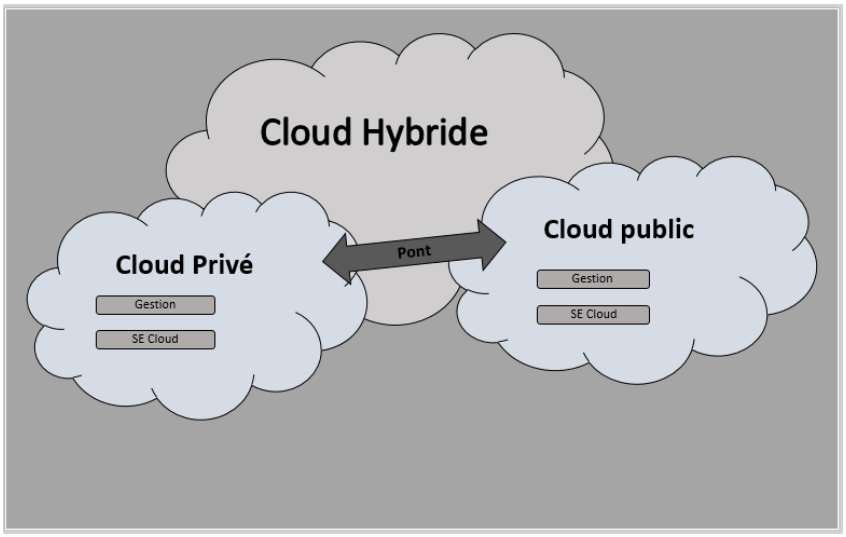

FIG. 1.3: Les types de Cloud computing.

1.6 Virtualisation

La virtualisation est un procédé qui permet de partager une seule instance physique d'une ressource ou d'une application entre divers clients et organisations. Elle est réalisée en suivant l'une des façons suivantes [10] :

- **Virtualisation au niveau du système d'exploitation** : dans cette virtualisation du Cloud computing, plusieurs instances d'une application peuvent s'exécuter à l'aide d'un seul système d'exploitation.
- **Virtualisation basée sur l'hyperviseur** : dans ce processus, le système d'exploitation partage le matériel de l'ordinateur hôte et par conséquent, il permet à plusieurs systèmes d'exploitation de s'exécuter sur un seul hôte.
- **Approche de la grille** : dans ce cas, une charge de travail donnée est distribuée à de nombreux serveurs physiques et une fois le résultat est calculé, il est renvoyé. Ce type de service est principalement utilisé à des fins scientifiques.

Les types les plus importants de virtualisation dans le Cloud computing sont comme suit [10] :

- **Virtualisation matérielle** : le gestionnaire de machine virtuelle (virtual machine manager ou VMM) s'installe en tant que logiciel dans le système matériel et par conséquent ce type de virtualisation est activé.
- **Virtualisation du système d'exploitation** : ici le VMM est installé dans le système d'exploitation d'un hôte plutôt que sur le matériel, ce qui permet de lancer plusieurs systèmes d'exploitation en même temps sur le même ordinateur.
- **Virtualisation de serveur** : est le fait de créer plusieurs serveurs virtuels sur un serveur physique. Ces serveurs créés tournent alors sur la même machine physique, tout en ayant les mêmes propriétés d'une machine physique.
- **Virtualisation du stockage** : le stockage disponible est virtualisé pour obtenir un grand accès au stockage virtuel, il est principalement utilisé pour fournir un processus de sauvegarde et de récupération.

La figure 1.4 représente le modèle de Cloud virtualisé.

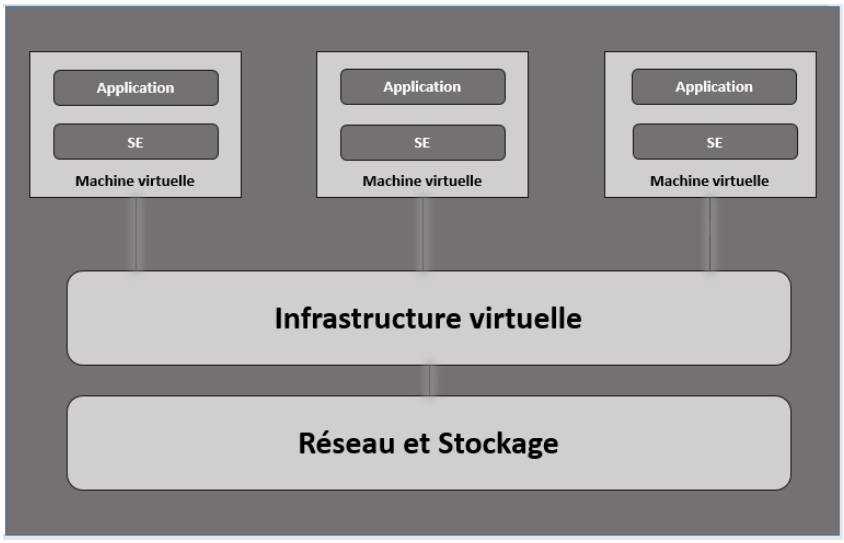

FIG. 1.4: Le modèle de Cloud virtualisé.

1.7 Avantages et inconvenients

1.7.1 Avantages

Le Cloud computing présente énormément d'avantages, on peut citer [11] :

— Pas de frais à l'acquisition.
— Aucun engagement de capital.
— Évolutif selon les besoins.
— Aucun personnel qualifié n'est nécessaire.
— Les centres de données sont parfaitement maintenus.

1.7.2 Limites

Le Cloud computing a beaucoup d'avantages. Toutefois, il a des points faibles qu'on peut citer dans ce qui suit [11] :

— Nécessite une forte et constante connexion internet.
— Préoccupations relatives à la protection des données personnelles.
— Dépendance vis-à-vis du fournisseur.
— Risque de sécurité pendant le transfert.
— Les prix bas incitent à réserver plus que ce dont vous avez besoin au total.

1.8 Conclusion

Dans ce chapitre, nous avons donné un aperçu global sur les notions de base du Cloud computing qui est le contexte de travail de notre PFE. Nous avons présenté la définition du Cloud, son architecture, ses types, la notion de virtualisation, les avantages du Cloud ainsi que ses inconvénients. Le chapitre suivant sera consacré à la présentation de l'approche adoptée pour la réalisation de notre travail.

Chapitre 2

Équilibrage de charge dans les systèmes distribués

2.1 Introduction

Les usages d'Internet se sont bouleversés d'une simple infrastructure de communication à une infrastructure de services électroniques plus complexe. De nos jours les internautes utilisent le Web aussi bien pour faire des recherches que pour effectuer des achats ou des transactions en ligne. Ces opérations entraînent une charge croissante pour les serveurs Web, qui sont responsables du fonctionnement convenable des boutiques en ligne, des portails d'information ainsi que des sites d'entreprises. Parallèlement au trafic sur le Web, les utilisateurs deviennent de plus en plus exigeants. Voulants minimiser les congestions et réclamants des temps de réponse infimes.

Malheureusement l'acquisition des techniques efficaces pouvant contrecarrer ce défaut chez les serveurs se révèle coûteuse. Cependant une autre alternative existe, qui consiste à répartir les charges de travail sur plusieurs serveurs Web. Celle-ci permet de contourner les pics de trafic mais aussi de sécuriser les sites Internet contre les pannes. Ce processus s'appelle le load balancing (ou en français, Équilibrage De Charge) [12].

2.2 Définition

L'équilibrage de charge est un procédé de lissage de trafic réseau, c'est-à-dire de répartir la charge globale vers différents équipements pour s'assurer de la disponibilité de ces derniers, en n'envoyant des données qu'à ceux en mesure de répondre, voire qui offrent le meilleur temps de réponse. Il permet d'améliorer sensiblement les performances des applications logicielles exécutées sur plusieurs serveurs en éloignant le trafic des requêtes client des serveurs saturés, ou en dysfonctionnement [13]. Cette technique d'optimisation est accomplie par un équilibreur de charge qui est un serveur virtuel qui reçoit les tâches et les assigne à des serveurs physiques, habituellement distribuées entre des serveurs répartis dans des centres de données géographiquement éloignés les uns des autres [14].

2.3 Approches d'équilibrage de charge

2.3.1 Statique vs dynamique

Dans une approche statique, les tâches sont assignées aux machines avant l'exécution de l'application qui les contient. Les informations concernant le temps d'exécution des tâches et les caractéristiques dynamiques des machines sont supposées connues. Les caractéristiques comme les vitesses des processeurs, les capacités des liens et le coût des calculs et des communications associés à chaque partition de données, si ils sont connues avec suffisamment de précision, alors un excellent niveau de performance peut être atteint par le biais de l'approche statique.

D'un autre côté une approche dynamique prend en compte la charge actuelle des nœuds dans un système distribué lors d'une répartition de tâches. S'il existe un changement important de charge pendant l'exécution, un transfert de tâches d'un nœud vers un autre est alors envisageable. L'algorithme utilisé dans ce type de distribution doit être capable de connaitre la charge des différents nœuds. Le but de ce transfert de tâches est de trouver un nœud, qui permettra d'alléger la charge sur un autre si un ensemble de tâches lui est soumis. Pour cela, il est nécessaire de connaître l'état de charge de tous les nœuds d'un système. Cette approche est beaucoup plus souple que l'approche statique, elle reste néanmoins approximative, car l'état global futur

d'un système ne peut être qu'estimé dans les meilleurs des cas. D'autre part, cette approche nécessite un surcotât de calcul pour analyser continuellement les états des nœuds, superviser l'échange inter-nœuds, calculer la charge de travail et la distribuer [15].

2.3.2 Centralisé vs distribué

Dans une approche centralisée, un site du système est choisi comme coordinateur. Il reçoit les informations de charge de tous les autres sites qu'il assemble pour obtenir l'état de charge global du système, et par conséquent dicter les décisions au reste.

Par contre dans le cas d'une approche distribuée, chaque site du système est responsable de collecter les informations de charge sur les autres sites et de les rassembler pour obtenir l'état global du système. Implicitement, les décisions de placement de tâches sont prises localement, étant donné que tous les sites ont la même perception de la charge globale du système [15].

2.3.3 Passive vs active

L'approche active dite source-initiative, survient lorsqu'un nœud surchargé (source) cherche à transférer une partie de sa charge vers un nœud (receveur) cible moins chargé. Si l'information de charge des nœuds est diffusée, alors le nœud source peut choisir un nœud cible en se basant sur cette information. Sinon, le nœud source doit envoyer des requêtes a plusieurs nœuds pour déterminer le nœud le plus approprié pour absorber son surplus de charge. Deux conditions doivent être satisfaites dans ce cas pour que le transfert de tâches soit effectif : la charge du nœud source doit dépasser le seuil admis et un nœud cible doit être trouvé.

On contraste l'approche passive, appelée receveur-initiative, est exécutée par un nœud non surchargé. Quand la charge d'un nœud est en-dessous du seuil minimal de charge, il demande à recevoir des tâches à partir des nœuds surchargés. Le transfert des tâches se fait par des méthodes similaires à celles utilisées dans les approches actives. L'approche active est plus performante quand la charge du système est basse ou moyenne, alors que l'approche passive est préférable quand la charge du système est grande [15].

Une stratégie hybride dite symétrique consiste à combiner les deux approches : active quand la charge du système est basse ou moyenne, et passive quand elle devient excessive.

2.4 Politiques d'équilibrage de charge

Dans le domaine d'équilibrage de charge, en parle de politiques et de mécanismes. Les politiques s'intéressent à l'ensemble des choix pour distribuer la charge, alors que les mécanismes effectuent physiquement la répartition de la charge et fournissent les informations exigées par les politiques [16].

La figure 2.1 montre les composants des politiques d'équilibrage de charge.

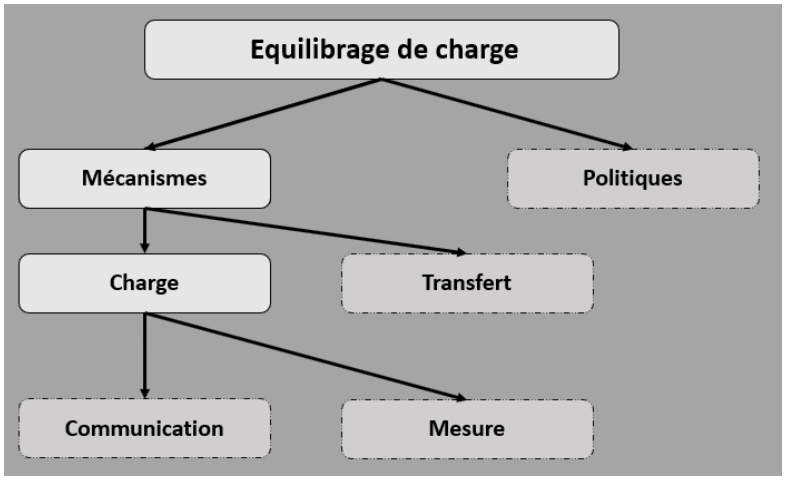

FIG. 2.1: Les composants des politiques d'équilibrage de charge.

2.4.1 Politique de participation

Le but de cette politique consiste à déterminer si un site est dans un état approprié pour participer à un transfert de tâches comme source (site surchargé) ou comme receveur (site sous-chargé), par conséquent partitionner l'ensemble des ressources en sources (surchargées), destinations (sous-chargées) ou neutres (équilibrées) [16].

2.4.2 Politique de sélection de la localisation

Cette politique est responsable de trouver, pour un site donné, un partenaire (source ou receveur), une fois que la politique de participation a décidé que ce site était soit source, soit receveur, le choix du partenaire se fait de manière aléatoire, ou en se basant sur les informations recueilli par le mécanisme de communication de la charge [16].

2.4.3 Politique de sélection du candidat

Une fois que les politiques de participation et de localisation ont décidé qu'un site « Si » est source et qu'un autre site « Sj » est receveur, cette politique est responsable du choix des tâches à transférer de « Si » vers « Sj ». Plusieurs méthodes de sélections sont à considérer [16] :

- Les tâches qui ont participé à la surcharge du nœud.
- Une sélection hasardeuse.
- Les tâches avec le temps de traitement le plus court.
- Les tâches avec le temps de traitement le plus long.
- …

2.5 Mécanismes d'équilibrage de charge

Les mécanismes de distribution de charge jouent trois grands rôles : le transfert de charge, la mesure de la charge et la communication de ses mesures [17].

La figure 2.2 représente les composants des mécanismes d'équilibrage de charge.

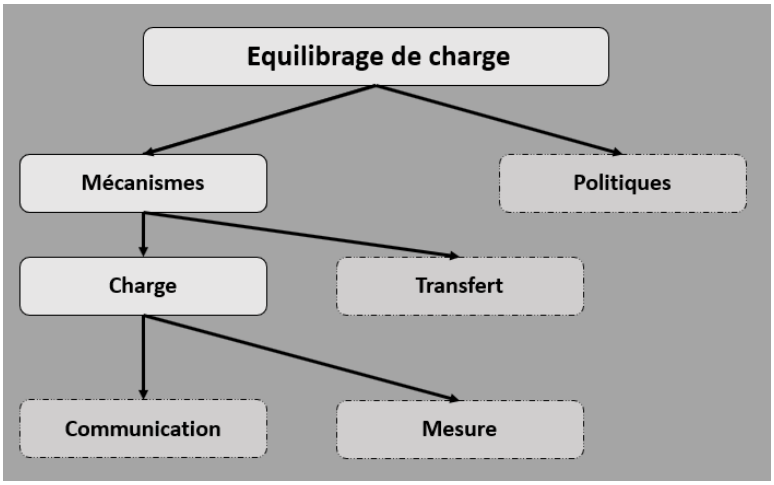

FIG. 2.2: les composants des mécanismes d'équilibrage de charge.

2.5.1 Le transfert de charge

Ce mécanisme est responsable du transfert physique de charge du nœud source vers le nœud destination. Ce transfert peut s'effectuer à deux moments : soit avant le début de l'exécution du processus sur le nœud source, on parlera alors de placement initial ; soit pendant l'exécution du processus sur le nœud source, on parlera alors de migration de processus [17].

2.5.2 La mesure de la charge

Le but de la mesure de charge est de pouvoir comparer la charge des nœuds et ce, en fonction de la politique choisie. La mesure de la charge consiste généralement en une combinaison d'indicateurs comme [17] :

— L'utilisation mémoire.
— La taille de la file d'attente du CPU.
— La taille de la file d'attente des périphériques E/S.
— Le nombre de processus exécutés.

— Le temps depuis la dernière activité sur les périphériques d'entrée (clavier, souris, etc.).
— Le temps de réponse après une sollicitation multicast.

2.5.3 La communication

Une fois les mesures de charge collectées localement, il faut les transmettre à l'agent central dans le cas d'une architecture centralisée, soit à tout ou à une partie des nœuds dans le cas d'une architecture distribuée. La difficulté principale vient du coût induit par la récolte et la distribution des informations de charge. En outre s'ajoute des problèmes de fiabilité et de consistance [17].

2.6 Quelques algorithmes standards d'équilibrage de charge

Dans cette section, nous présentons deux algorithmes standards d'équilibrage de charge.

2.6.1 Round robin

— **Principe** : cet algorithme est très simple, la première requête est envoyée au premier serveur, puis la suivante au second, et ainsi de suite jusqu'au dernier. Ensuite on recommence en attribuant la prochaine requête au premier serveur, etc.

— **Implémentation** : Round Robin est une procédure aller-retour dans laquelle les demandes de serveurs entrants sont gérées par l'équilibreur de charge dans une file d'attente et distribuées aux serveurs connectés en série. La méthode round-robin traite tous les processus de la même manière, indépendamment de l'urgence de la requête ou de la charge du serveur qu'elle provoque. Un répartiteur de charge fonctionnant selon le principe round-robin est donc particulièrement adapté aux environnements dans lesquels environ ou les mêmes ressources sont disponibles pour tous les serveurs du cluster [12].

— **Caractéristiques :**
o Largement utilisé puisque facile à mettre en œuvre.
o Si les serveurs ont des capacités de traitement différentes, certain peuvent devenir surchargés et planterons [18].
o Ce type d'EDC est trop limité.
o Ne prend pas en considération les capacités des serveurs.
o Risque élevé de congestionner un serveur avec de faible capacité par des requêtes lourdes par conséquent des requêtes facile à traiter sont dirigées vers un serveur robuste [19].

La figure 2.3 représente la gestion des tâches en Round Robin.

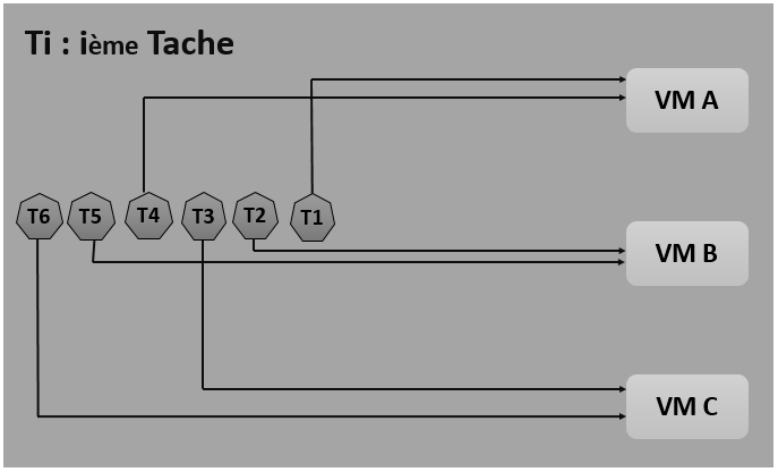

FIG. 2.3: la gestion des tâches en Round Robin.

2.6.2 Least Connection

— **Principe et implementation** : répartit les demandes en fonction des connexions existantes du serveur respectif : celui qui a le plus petit nombre de connexions actives reçoit la demande suivante de l'équilibreur de charge. Cette méthode d'équilibrage de charge est recommandée pour les clusters de serveurs homogènes où des ressources comparables sont disponibles pour tous les ordinateurs. Le fait de ne pas le faire peut entraîner des retards dans la réponse aux

demandes [18].

— **Caractéristiques :**
o Empêche la surcharge d'un serveur en vérifiant le nombre de connexions du serveur.
o Ne tient pas compte de la capacité du serveur lors de la détermination du nombre de connexions actuelles, c.-à-d. si le serveur A a 10 connexions et le serveur B en a 20. La prochaine demande sera envoyée au serveur A vu qu'il en a moins, mais en fait si le serveur A peut avoir une capacité de 12 connexions. Tandis que le serveur B en a 50, rendra le serveur A plus susceptible de devenir surchargé.

La figure 2.4 représente la gestion des tâches en Least Connection.

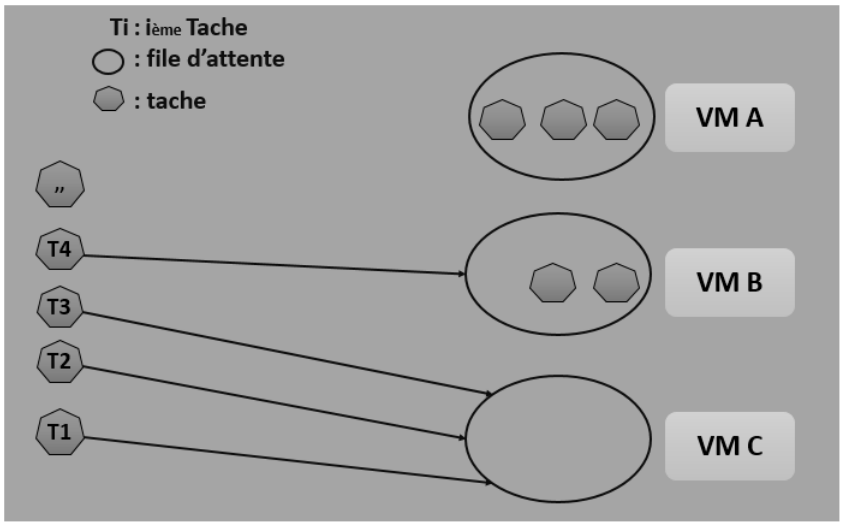

FIG. 2.4: la gestion des tâches en Least Connection.

L'algorithme qui a été évalué dans le cadre de notre projet de fin d'études sera présenté en détail dans le chapitre 3.

2.7 Indices de performance

Parmi tous les indices proposés dans la littérature, nous pouvons retenir les suivants [19] :

- Du point de vue de l'utilisateur, l'indice le plus important est le temps de réponse, c'est-à-dire le temps qui s'écoule entre l'instant où la tâche est soumise au système et l'instant où celle-ci est complétement exécutée.
- Du point de vue des ressources, l'indice mesuré est le nombre de tâches traitées par unité de temps par le système distribué.
- Le nombre de tâches présentes à un instant t dans le système représente la charge de travail instantanée de ce système. Cet indice caractérise aussi les performances d'un système.
- Le nombre de communications effectuées dans le système pendant une unité de temps, appelé taux de communication, est mesuré pour minimiser le surcout dû aux transferts.

L'efficacité d'un système de répartition de charge peut être déterminée par la prise en charge de tous ces indices de performances qui sont parfois en dualité, ce qui nécessite de satisfaire les propriétés suivantes [19] :

- **Efficacité** : Un système d'équilibrage de charge doit pouvoir exploiter la puissance maximale d'une ressource en minimisant son temps d'inactivité. Il est donc possible de vérifier son efficacité en observant directement la proportion du temps d'inactivité pour chaque ressource.

- **Equité** : Elle consiste à attribuer, a un instant donné, le même temps d'exécution à des tâches ayant les mêmes priorités. Sur une ressource donnée, cette propriété est assurée par l'ordonnanceur. Cependant, celui-ci n'a pas les moyens d'ajuster le temps alloué à chaque tâche en fonction de ce qui se passe sur les autres ressources. C'est donc le système d'équilibrage de charge qui devra assurer l'équité inter-ressources.

- **Localité** : Les tâches qui ont une affinité particulière pour une ou plusieurs ressources doivent y être affectées en priorité. Cette propriété implique en général une réduction du nombre de migration et aussi, pour une tâche donnée, un ratio de temps d'exécution

supérieur sur certaines ressources que sur d'autres.

— **Stabilité** :Il faut veiller à ce que toutes les tâches s'exécutent en un temps fini. Nous devons prêter attention au phénomène de va-et-vient perpétuel (ping-pong) entre les nœuds qui interviennent dans l'équilibrage de charge.

— **Extensibilité** : Les performances d'un système d'équilibrage ne doivent pas se dégrader avec l''augmentation du nombre de nœuds dans le système.

— **Transparence** : Le système d'équilibrage doit s'exécuter de manière transparente par rapport à l'utilisateur. Ce dernier ne doit pas prendre part à la politique de la répartition de charge, sauf dans des cas extrêmes et bien spécifiques.

2.8 Conclusion

Ce deuxième chapitre a été dédié à la présentation de l'équilibrage de charges dans les systèmes distribués. Nous avons donc présenté ses approches, ses politiques, ses mécanismes, quelques algorithmes standards pour sa mise en œuvre ainsi que les indices de performance utilisés dans ce domaine. Le dernier chapitre de ce rapport sera consacré à la présentation de notre contribution dans le cadre de ce projet de fin d'études. Cette contribution concerne l'implémentation et l'évaluation d'un algorithme d'équilibrage de charges appliqué au Cloud computing. L'évaluation a été faite en utilisant CloudSim Plus.

Chapitre 3

Implémentation et évaluation de l'algorithme d'équilibrage de charges

3.1 Introduction

Ce chapitre a pour objectif d'exposer l'algorithme d'équilibrage de charges que nous avons implémenté dans le cadre de ce PFE. Nous allons commencer par la présentation de cet algorithme ainsi que sa terminologie. Par la suite, nous présentons les divers outils utilisés pour le développer et l'implémenter, notamment le simulateur CloudSim Plus qui a été utilisé pour tester et simuler cet algorithme dans un environnement Cloud. Les résultats obtenus sous forme de graphes sont présentés également dans ce chapitre.

3.2 Présentation de l'algorithme

3.2.1 Description

L'algorithme que nous avons implémenté et évalué dans le cadre de ce PFE est un nouvel algorithme distribué pour les charges indivisibles dans les réseaux arbitraires. Ce dernier est inspiré du problème de sacs à dos multiple en considérant le réseau des nœuds comme de sacs à dos similaires, tous remplis de certains éléments de charge, chacun avec un poids qui représente une charge indivisible dans notre algorithme. L'objectif est de déterminer le

nombre d'articles à attribuer dans chaque sac à dos afin que le poids total soit inférieur ou égal à sa capacité jusqu'à ce que l'équilibre soit atteint entre tous les sacs à dos.

Soit G = (N ; E) un graphe connecté représentant un réseau arbitraire, où N est l'ensemble des nœuds et E est les liens entre les nœuds. Chaque nœud de N n'a qu'une vue partielle du système. Dans notre étude, nous considérons le cas de charges sporadiques non divisibles pouvant arriver sur n'importe quel nœud a n'importe quel moment et rivaliser pour les ressources de calcul du réseau. Nous nous concentrons sur l'exclusion mutuelle par paire d'équilibrage (Pair-Wise). Le terme d'exclusion mutuelle vient du fait qu'à chaque itération, nous pouvons avoir en parallèle des paires indépendantes de nœuds voisins impliqués dans le processus d'équilibrage.

3.2.2 Pseudo code

Dans notre algorithme, après la détection d'un déséquilibre par un nœud avec l'un de ses voisins (différence de charges entre les deux nœuds selon un seuil), les sept règles suivantes sont déclenchées.

L'algorithme va être décrit du point de vue d'une paire de voisins $(i ; j) \in N$.

Algorithme d'équilibrage de charges indivisibles

R1: Invitation

Un nœud envoie une invitation à l'un de ses voisins avec lequel il a détecté un déséquilibre à condition qu'il n'ait pas en cours de transaction avec un autre nœud. En effet, le nœud doit vérifier qu'il n'a pas déjà envoyé ou accepté une invitation.

R2: Acceptation

Pour que l'invitation d'un voisin soit acceptée, le nœud invité ne doit pas être en transaction avec un autre voisin.

R3: Transaction

Une fois l'invitation envoyée et acceptée, la transaction peut commencer en appelant la méthode Pair-Wise() qui sera détaillée dans ce qui suit.

R4: Valider la transaction

La validation de la transaction se fait par cette règle. Dans ce cas, les charges virtuelles d'un nœud deviennent effectives afin de minimiser le nombre de migrations inutiles entre les nœuds.

R5: Libérer le pointeur après la transaction

Une fois la transaction terminée, le nœud qui a envoyé l'invitation doit réinitialiser ses variables. Ça sera pareil pour celui qui a reçu l'invitation.

R6: Correction du pointeur d'invitation

Retirer l'invitation en cas d'absence de déséquilibre entre un nœud et ses voisins.

R7: Correction du pointeur d'acceptation

Retirer l'acceptation en cas d'absence de déséquilibre entre un nœud et ses voisins.

Le pseudo code basique de la fonction Pair-Wise() est comme suit :

Fonction : Pair-Wise

Variables :
- F(n) : liste de fusion.
- n : charge de la tâche.

Listes :
- N(i) : liste des voisins.
- L(i) : liste des charges (tâches).
- V(i) : liste virtuelle des charges (tâches).

Initialisation :
- F(n) = L(i) ∪ L(j)
- V(j) = ∅
- L(i) = ∅

while F(n) ≠ ∅ **do**
    if (i.VirtualTotalLoad ≤ j.VirtualTotalLoad) **then**
        L(i) = L(i) ∪{n}
    else
        V(j) = V(j) ∪{n}
    End if
    $F(n) = F(n) - \{n\}$
End while

3.3 Environnement de développement et de simulation

3.3.1 Environnement de développement

— **Java** : C'est un langage orienté objets développé par Sun Microsystems (aujourd'hui racheté par Oracle). Le nombre d'applications conçues avec ne cesse de croître et cela montre la robustesse de ce dernier. De plus, nous avons a opté pour Java car il est rapide, sécurisé et fiable. Des ordinateurs portables aux centres de données, des consoles de jeux aux superordinateurs scientifiques, des téléphones portables à Internet, la technologie Java est présente sur tous les fronts [20].

— **JavaFx** : Oracle a décidé de faire de JavaFX un outil distinct de Java, et de le distribuer sur le site OpenJFX. JavaFX veut être le framework des applications Web ainsi que des applications sur mobile. Cela le met en concurrence avec plusieurs framework déjà populaires ainsi qu'avec Android, plateforme de développement, et OS mobile de Google. Il permet de réaliser plus facilement des interfaces que Swing, avec un design moderne et convivial [21].

— **Maven** : Un outil permettant de gérer les dépendances de notre projet et d'automatiser sa construction (compilation, test, production de livrable...). Il offre entre autres les fonctionnalités suivantes [22] :
  — Compilation et déploiement des applications Java (JAR, WAR).
  — Gestion des librairies requises par l'application.
  — Exécution des tests unitaires.
  — Génération des documentations du projet (site web, pdf, Latex).
  — Intégration dans différents IDE (Eclipse, JBulder).

— **IntelliJ** : La plateforme IntelliJ est une plateforme libre développée par JetBrains pour créer des environnements de développement intégré et des outils de développement prenant en charge les langues. Elle est utilisée par IntelliJ IDEA, Android Studio et Cursive pour n'en nommer que quelques-uns. Basée sur Java, elle offre une approche multiplateforme de la création d'outils pour tout type de langage,

qu'il cible ou non la JVM [23].

— **SceneBuilder** : Un outil initialement créé par Oracle et désormais faisant partie de l'OpenJFX qui permet de créer des fichiers au format FXML via un éditeur graphique. Cet outil est disponible en tant qu'application autosuffisante qui peut être lancée depuis votre bureau ou en tant qu'API intégrable dans des outils tiers tels que IntelliJ et NetBeans [24].

3.3.2 Environnement de simulation : CloudSimPlus

CloudSimPlus est un simulateur moderne, complet et entièrement documenté. Il est facile à utiliser et à étendre, permettant la modélisation, la simulation et l'expérimentation des infrastructures de Cloud computing et des services d'application. Il permet aux développeurs de se concentrer sur des problèmes spécifiques de conception de système à étudier, sans se soucier des détails de bas niveau liés aux infrastructures et services basés sur le Cloud. CloudSim Plus est basé sur CloudSim 3, le simulateur moderne de Cloud, hautement extensible et facile à utiliser. Les changements apportés à ce dernier visent à permettre aux projets une évolution durable et une maintenabilité à long terme [25].

CloudSim Plus a une structure plus simple pour en faciliter l'utilisation et la compréhension. Il se compose de 4 modules, dont deux nouveaux [25].

Figure 3.1 représente les différents modules de CloudSim Plus.

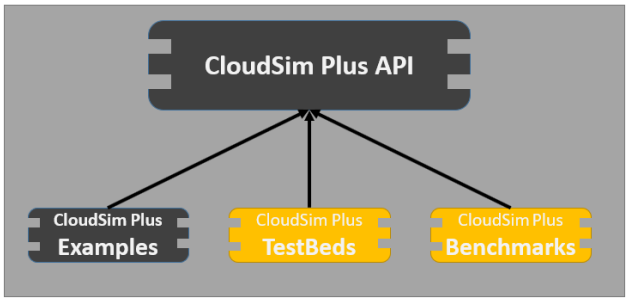

FIG. 3.1: les différents modules de CloudSim Plus.

L'API CloudSim Plus est le module principal qui représente l'API du simulateur, un module indépendant et unique pour la construction de simulations.

— **Examples CloudSim Plus** : exemples exclusifs pour refactoriser les précédents.
— **TestBeds CloudSim Plus** : simulations à plusieurs étapes pour une collecte de données scientifiquement valide.
— **Benchmarks CloudSim Plus** : évaluation des performances de fonctionnalités telles que l'heuristique.

La figures 3.2 représente la structure du package de l'API CloudSim Plus.

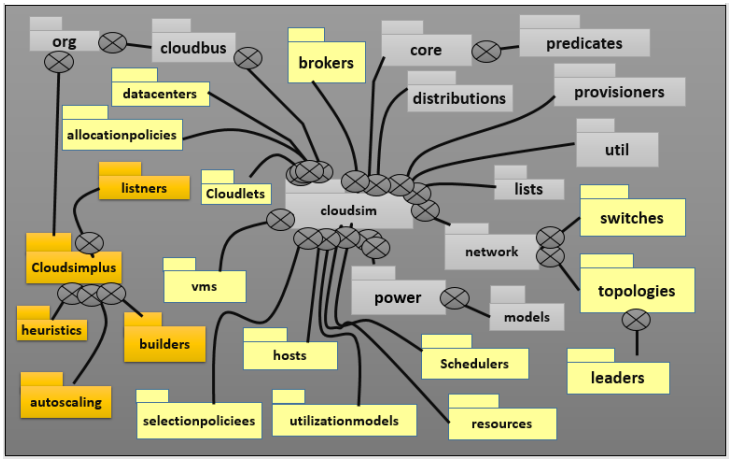

FIG. 3.2: la structure du package de l'API CloudSim Plus.

— **Orange :** les packages CloudSim Plus avec des caractéristiques exclusives.
— **Jaune claire :** de nouveaux packages pour mieux organiser les classes CloudSim.
— **Gris :** les package déjà existants dans CloudSim.

Dans ce PFE et pour la simulation de notre algorithme, nous avons préféré utiliser CloudSimPlus à la place de CloudSim et cela pour différents raisons [25] :

— Il est plus simple à utiliser.
— Il permet de retarder la création des VMs et des Cloudlets soumis, permettant ainsi la simulation de l'arrivée dynamique des tâches.

— Il permet d'allumer et d'éteindre automatiquement les hôtes en fonction de la demande.
— Il intègre une classe *DatacenterBrokerSimple* fonctionnelle qui permet de modifier, au moment de l'exécution, les politiques pour différents objectifs. Ce comportement dynamique permet d'implémenter des politiques spécifiques sans nécessiter la création de nouvelles classes *DatacenterBroker*.
— Il permet la création d'hôtes au moment de l'exécution de la simulation pour permettre l'expansion physique de la capacité du centre de données.
— Il permet une allocation de ressources à la demande pour les VMs (RAM, bande passante et CPU).
— Il permet la création dynamique de VMs en fonction d'une condition de surcharge. Une telle condition est définie par un prédicat qui peut vérifier l'utilisation de différentes ressources.
— Il permet un calcul de la consommation d'énergie avec une grande précision pour différents intervalles de planification du Datacenter.
— Il permet un calcul intégré de l'historique d'utilisation du processeur et de la consommation d'énergie pour les VMs (et les hôtes).

3.4 Topologies logiques utilisées

L'algorithme qui a été implémenté dans le cadre de ce PFE est un algorithme distribué qui nécessite lors de son exécution des paires de voisins indépendants impliqués dans le processus d'équilibrage. La notion de voisinage doit être définie et par conséquent des topologies logiques doivent être utilisées. Dans ce qui suit, nous avons opté pour les topologies suivantes :

— Au sein d'un même hôte : la topologie en étoile.
— Entre les hôtes / DataCenters : la topologie en anneau. Un DataCenter contient plusieurs hôtes et un hôte contient plusieurs VMs.

La figure 3.3 représente un exemple de topologies.

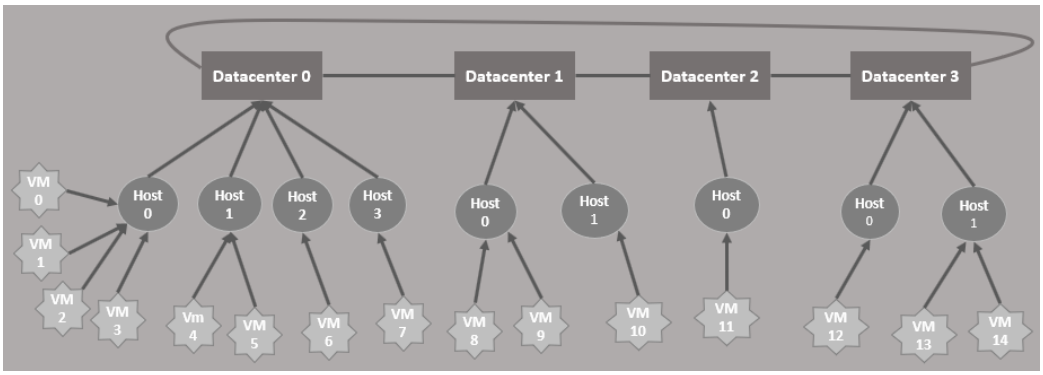

FIG. 3.3: Exemple de topologies.

Le tableau 3.1 représente les voisins de chaque VM dans la topologie précédente.

Machine virtuelle	Liste des voisins
VM0	1-2-3-4-7-8-12
VM1	0-2-3
VM2	0-1-3
VM3	0-1-2
VM4	5-0-6
VM5	4
VM6	4-7
VM7	0-6
VM8	9-10-0-11
VM9	8
VM10	8
VM11	8-12
VM12	13-11-0
VM13	14-12
VM14	13

TAB. 3.1: Tableau des voisins.

Par exemple, les voisins de Vm 0 sont : Vm 1, Vm 2, Vm 3, Vm 4, Vm 7, Vm 8 et Vm 12. Vm 1, Vm 2 et Vm 3 car ils sont au sein du même hôte qui est le Host 0. Vm 4 de Host 1 et Vm 7 de Host 3 parce que Host 0 et voisin avec Host 1 et Host 3, et il n'y a que le premier Vm de chaque hôte qui peut avoir

un voisinage avec les Vms des autres Hosts. Vm 8 de Host 0 du Datacenter 1 et Vm12 du Host 0 du Datacenter 3 puisque le Datacenter 0 est voisin avec Datacenter 1 et Datacenter 3, et il n'y a que le premier Vm de l'Host 0 qui peut avoir un voisinage avec les Vms des autres Datacenters.

3.5 Présentation de l'application

— **Interface principale :** Une fois l'application lancée, la fenêtre illustrée dans la figure 3.4 apparait, on trouve dans cette dernière :
o Une barre de menu détaillée facilitant la manipulation de la persistance des configurations.
o Un volet d'onglets regroupant les fonctionnalités principales de cette dernière à savoir : L'onglet Cloud configuration, Model configuration, Tables, Graphical visualization et Obtained results.
o Un log pour rappeler les diverses séries d'étapes déjà faites, et afficher des commentaires facilitant la compréhension du déroulement de certaines tâches.
Figure 3.4 représente l'interface principale de notre application.

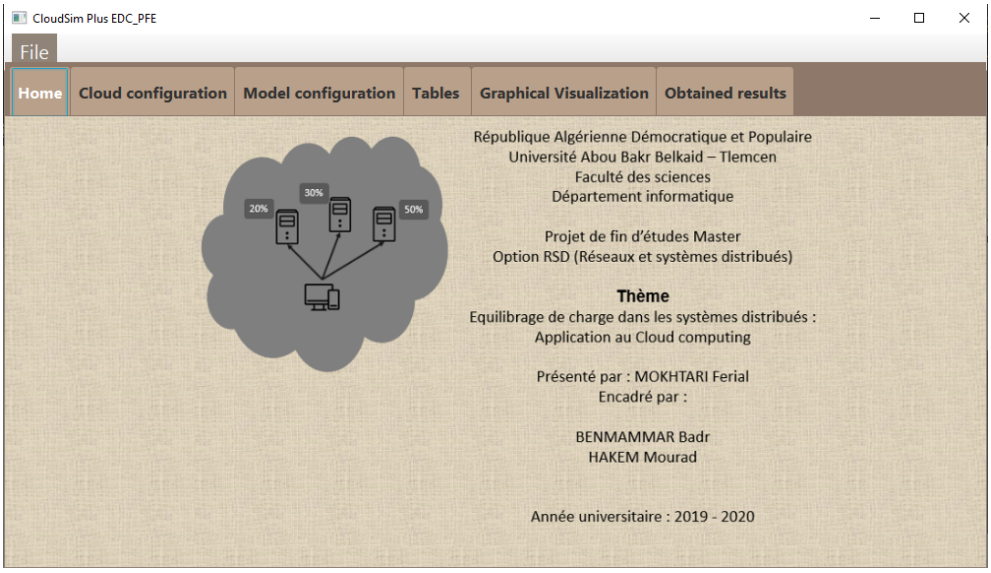

FIG. 3.4: L'interface principale de l'application.

L'interface principale permet de créer un Cloud en ajoutant un ou plusieurs DataCenters, un ensemble d'hôtes et de machines virtuelles.

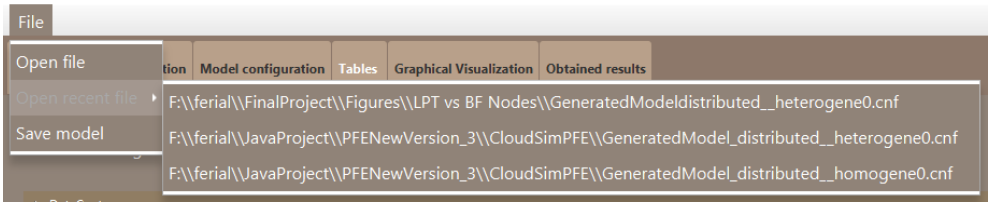

FIG. 3.5: Le menu.

Ce menu comporte les choix suivants :
o **Open file :** permet de paramétrer une nouvelle simulation en ouvrant un fichier d'une configuration déjà enregistrée lors d'une simulation antérieur.
o **Open recent file :** pour accéder aux configurations ouvertes récemment afin de les charger sans avoir à spécifier le chemin.
o **Save model :**permet d'enregistrer le modèle de la configuration paramétrée pour l'utiliser dans la production des statistiques.

— **Configuration du Cloud :**
o **Ajout des datacenters :** la figure 3.6 représente l'interface d'ajout des DataCenters.

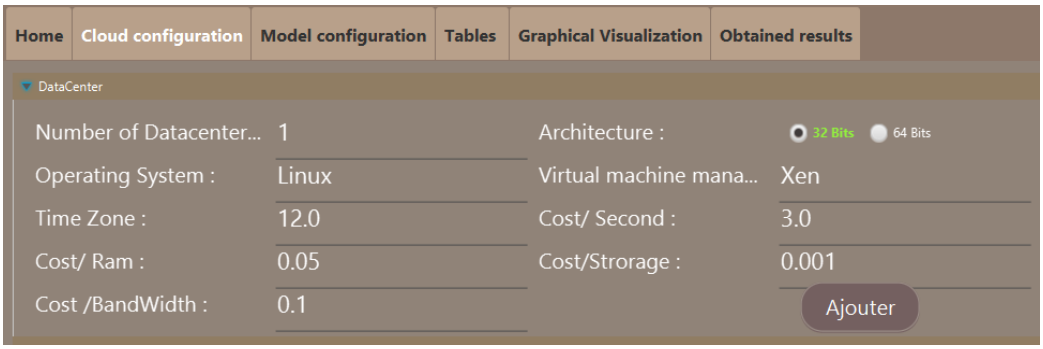

FIG. 3.6: l'interface d'ajout des datacenters.

Cette interface permet de spécifier le nombre de DataCenters,

ainsi que leurs caractéristiques. Après avoir spécifié toutes les caractéristiques, et en cliquant sur Ajouter, les nouveaux DataCenters sont ajoutés.

o **Ajout des Hôtes :** la figure 3.7 représente l'interface d'ajout des hôtes.

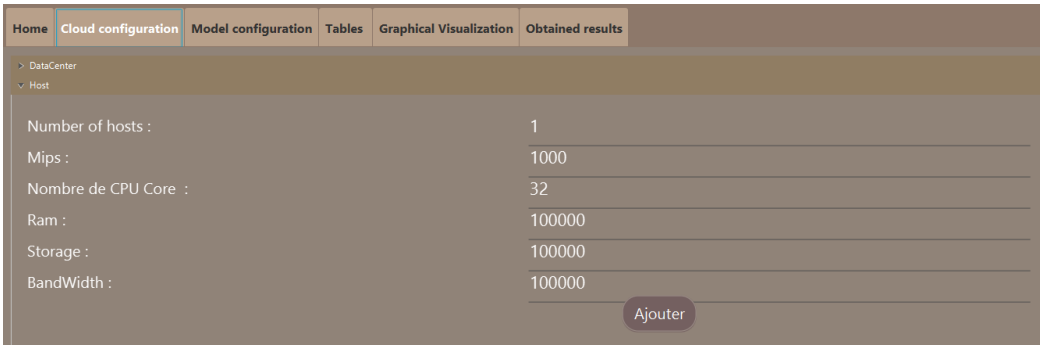

FIG. 3.7: l'interface d'ajout des hotes.

Cette interface permet de spécifier le nombre de Host ainsi que leurs caractéristiques. Aprés avoir spécifié toutes les caractéristiques, et en cliquant sur Ajouter, les machines hôtes sont ajoutées.

o **Ajout des machines virtuelles :** la figure 3.8 représente l'interface d'ajout des machines virtuelles.

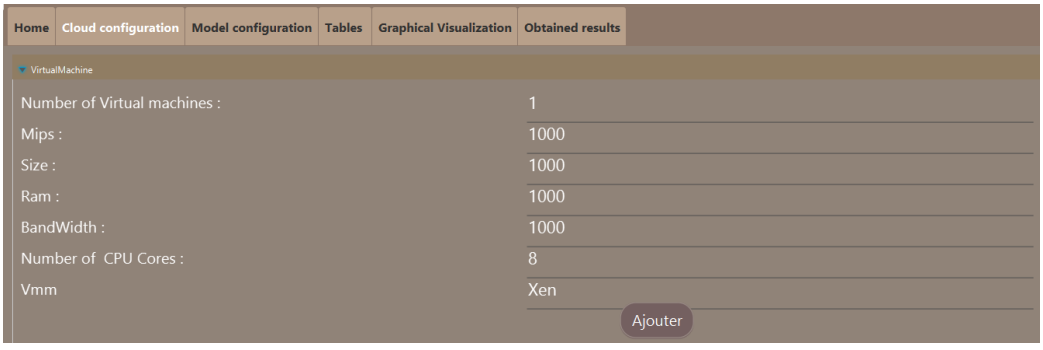

FIG. 3.8: l'interface d'ajout des machines virtuelles.

Cette interface permet de spécifier le nombre de machines virtuelles,

ainsi que leurs caractéristiques. Après avoir spécifié toutes les caractéristiques, en cliquant sur Ajouter, les machines virtuelles seront ajoutées.

o **Ajout des tâches :** la figure 3.9 représente l'interface d'ajout des tâches.

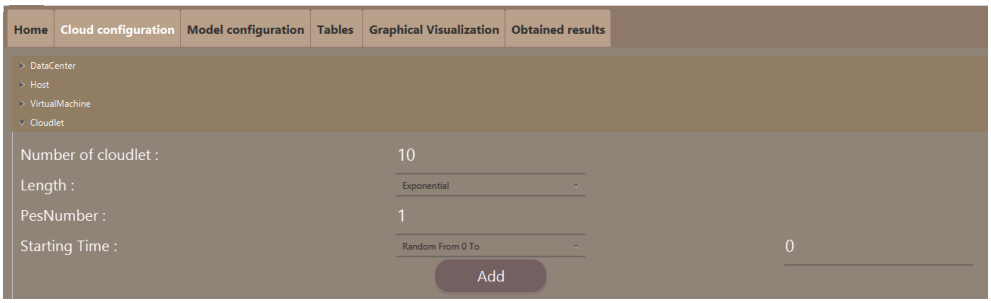

FIG. 3.9: l'interface d'ajout des tâches.

Cette interface permet de spécifier le nombre de tâches (cloudlets), ainsi que leurs caractéristiques. Après avoir spécifié toutes les caractéristiques, et en cliquant sur Ajouter, les tâches seront ajoutées.
o **Log :** la figure 3.10 représente un exemple de messages affichés dans le log.

FIG. 3.10: Exemple de messages affichés dans le log.

L'utilisateur peut voir des différents messages de chaque action (l'ajout d'un Dc, Vm ou Hôte… l'enregistrement d'un modèle ou bien la fin d'une simulation, etc.).
Comme il peut le vider dans le menu contextuel avec un clic droit sur la souris, puis en cliquant sur Clear.

— **Configuration d'un model :** permet de configurer un modèle de simulation : nombre de DCs, de Hosts et de Vms ainsi que le choix

de configuration pour les VMs (homogène ou hétérogène). Après avoir cliqué sur le bouton « **Generate model** », les composants seront crées et enregistrés sous forme d'un nouveau fichier avec l'extension . cnf, et avec le préfix adéquat.
La figure 3.11 représente l'interface de configuration d'un modèle de simulation.

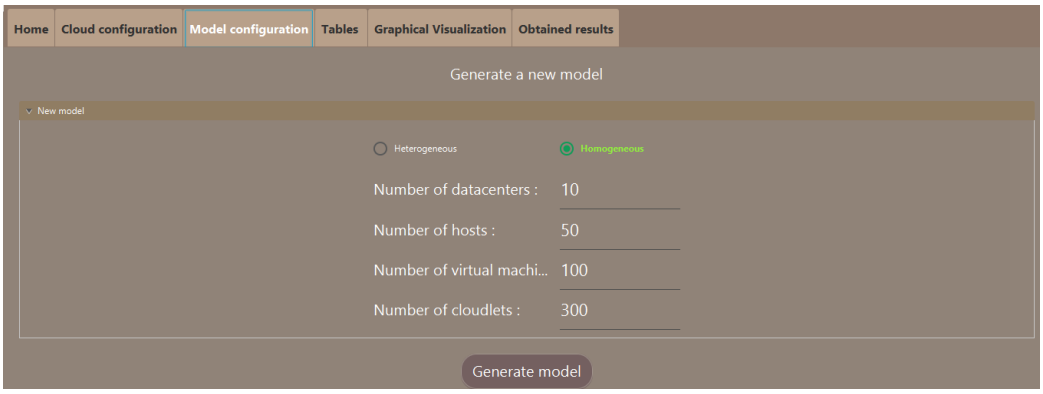

FIG. 3.11: L'interface de configuration d'un model de simulation.

— **Tableaux :** Après avoir créé une configuration au niveau du premier onglet (Cloud configuration) ou bien charger un modèle depuis le menu, les tableaux ci-dessous se remplissent. Par conséquent, nous pouvons consulter le tableau des DataCenter, des Hôtes, des machines virtuelles et des Cloudlets ajoutées.
Les tables suivantes sont des exemples des divers tables une fois remplîtes.

DataCenter

Name	Architecture	Vmm	OS	TZ	Cost Per Stora...
dc0	x86	Xen	Linux	10.0	0.001
dc1	x86	Xen	Linux	10.0	0.001
dc2	x86	Xen	Linux	10.0	0.001
dc3	x86	Xen	Linux	10.0	0.001

TAB. 3.2: Tableau des datacenters.

Host

ID Host	DataCenter	Ram	Storage	BW
0	dc0	1000000	100000000	1000000
0	dc1	1000000	100000000	1000000
0	dc2	1000000	100000000	1000000
0	dc3	1000000	100000000	1000000

TAB. 3.3: Tableau des hotes.

Vm

ID	Host ID	Mips	Storage	Ram	BW
0	-1	1000.0	100000	2048	100000
1	-1	1000.0	100000	2048	100000
2	-1	1000.0	100000	2048	100000
3	-1	1000.0	100000	2048	100000

TAB. 3.4: Tableau des machines virtuelles.

Cloudlet

Cloudlet ID	Delay To Start	Length	Pes Number
0	9.0	77003	1
1	1.0	37766	1
2	2.0	51158	1
3	5.0	48485	1

TAB. 3.5: Tableau des tâches.

— **Simulation :**
Avant la création du Cloud, le bouton de simulation était désactivé. Après la détection d'un nouveau Cloud, le bouton sera activé est la simulation peut être lancée en choisissant l'algorithme comme suit :

FIG. 3.12: Interface avec le choix RR (round-robin).

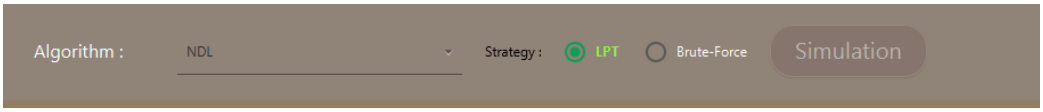

FIG. 3.13: Interface avec le choix NDL (Non Divisible Loads).

Le choix NDL propose les deux stratégies suivantes :

- NDL-LPT : dans la stratégie LPT (Largest Processing Time first), les tâches seront triées par ordre décroissant de leurs longueurs, pour ensuite les assigner consécutivement dans cet ordre, à la machine la moins chargée entre les deux. Cette stratégie, donne une solution approchée (en local).

- NDL-BF : dans la stratégie BF (Brute Force), une recherche exhaustive est faite avec une complexité de $O(2^N)$ (N : Nombre de tâches) pour avoir a la fin deux listes de tâches avec un écart de longueur minimal. Cette stratégie, donne une solution optimale (toujours en local).

L'objectif des deux stratégies est de se converger vers une solution optimale globale sachant que le BF explose en termes de temps d'exécution sur des instances de grande taille.
Lorsque l'utilisateur lance la simulation il se trouvera devant le message suivant :
o Oui : Pour enregistrer la configuration sous forme d'un fichier .cnf.
o Non : Pour lancer la simulation sans faire de sauvegarde.
o Annuler : Pour annuler la simulation et effectuer des retouches sur la configuration.

La simulation est lancée et une fois terminée, les résultats sont

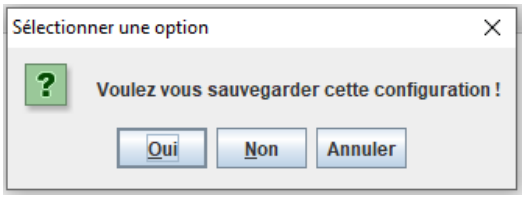

FIG. 3.14: Message de sauvegarde.

visualisés dans le tableau « Result » comme suit (voir le tableau 3.6) :

Result

cloudletId	DataCenter	HostId	vmId	Lenght	status	time
90	dc4	0	14	10138	SUCCESS	10.248
81	dc4	1	9	12024	SUCCESS	12.243999999
26	dc2	0	2	13410	SUCCESS	13.630000000
28	dc0	1	5	13856	SUCCESS	13.966
30	dc1	0	1	14077	SUCCESS	14.187
22	dc4	0	14	14761	SUCCESS	14.981000000

TAB. 3.6: Tableau de résultat.

Le bouton « Advanced Statisics » dans l'onglet « Obtained results » permet d'afficher une IHM dans laquelle on peut mesurer les performances de notre algorithme à travers divers métriques comme : le makespan, le nombre de migrations, le temps de réponse moyen, le temps d'inactivité moyen et l'écart de la solution fournie par notre algorithme par rapport la solution idéale.

La figure 3.15 représente l'IHM de « Advanced Statisics ».

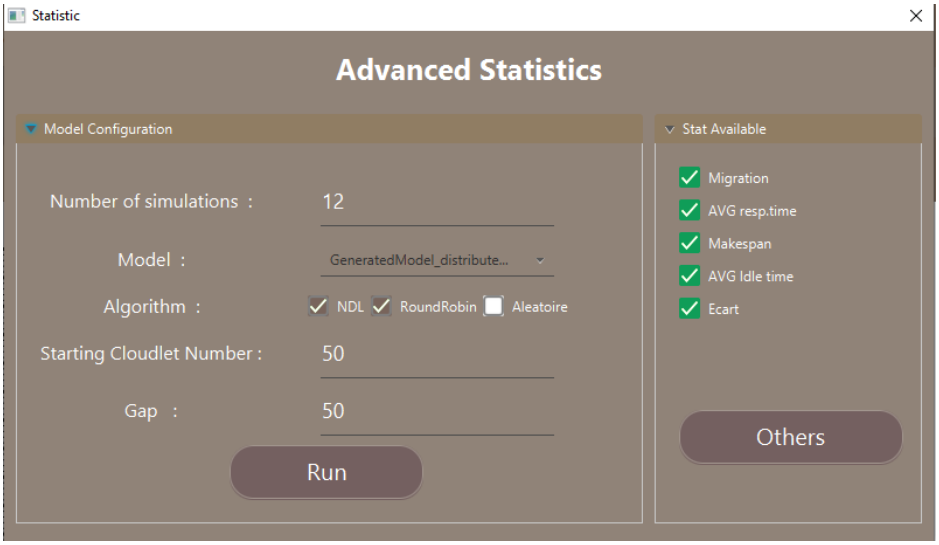

FIG. 3.15: l'IHM des statistiques avancées.

Il est impératif de remplir ces diffèrent champs :

— La configuration cible.
— La plage des Cloudlets : Start Cloudlet Number, Gap, Number of simulations.
— Les algorithmes à traiter.
— Les métriques à évaluer.

On peut par la suite accéder à d'autres types de métriques en cliquant sur le bouton « Other Statistics », et on aura une autre interface avec d'autres métriques, comme le montre la figure 3.16.

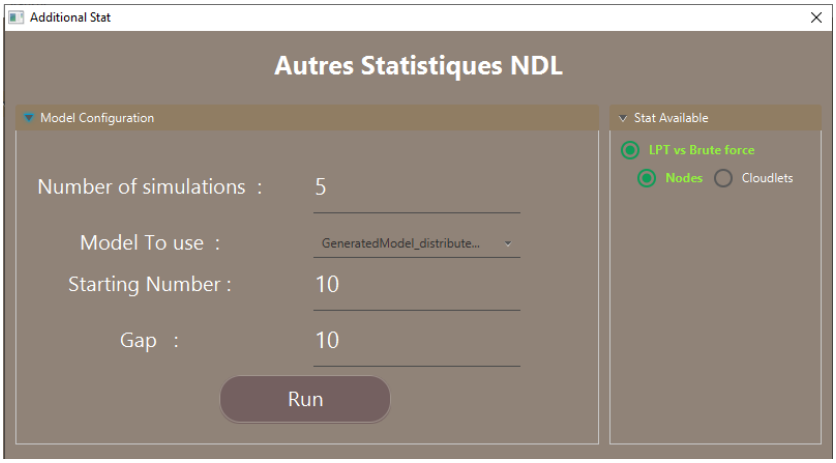

FIG. 3.16: L'IHM pour d'autres statistiques.

— **Visualisation graphique :** permet de visualiser notre configuration sous forme d'un réseau contenant les DataCenters, les hôtes, les Vms ainsi que les emplacements finaux des Cloudlet en utilisant la bibliothèque VisFX [26] (voir la figure 3.17).

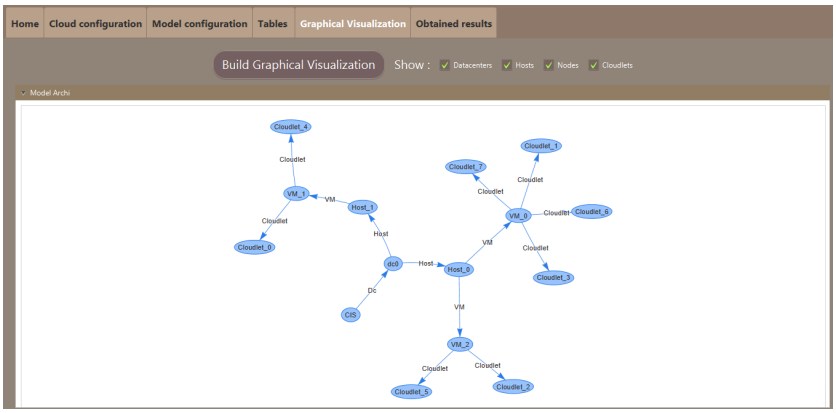

FIG. 3.17: Exemple d'un réseau distribué.

Dans la figure précédente, nous avons préféré montrer un petit nombre de nœuds juste par soucis de lisibilité de la figure, sinon, nous pouvons faire une visualisation peu importe la taille du réseau.

3.6 Évaluation des résultats obtenus

3.6.1 Dans un environnement homogène et avec des instances de petite taille

Dans ce qui suit, nous présentons l'évaluation de notre algorithme dans un environnement homogène et avec une configuration minimaliste. Notre configuration contient 5 Datacenters, 10 hôtes et 20 Vms. Les hôtes et les VMs sont distribués sur les datacenters selon le Round Robin. Les VMs sont caractérisées par : 4 cores chacune, une capacité de RAM égale à 2048 MB et une capacité de bande passante égale à de 100 MB/s. Nous avons simulé l'arrivée de 500 tâches d'une longueur variable (exponentielle) et générée d'une manière aléatoire entre 10 000 et 80 000 MI avec des temps d'arrivées aléatoires des tâches entre 0 et 10 secondes.

Nous avons calculé la charge des différentes VMs après l'exécution de notre algorithme (NDL-LPT), l'exécution du Round-Robin et le cas sans traitement (l'arrivée aléatoire des tâches).
La figure 3.18 représente les résultats obtenus.

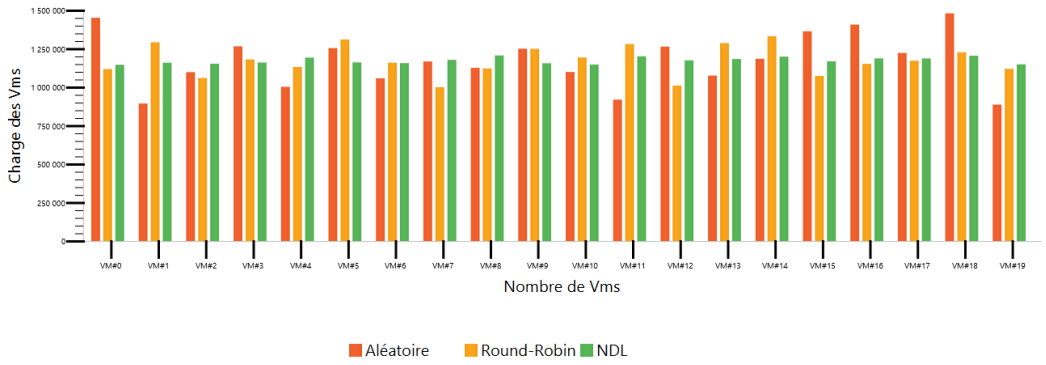

FIG. 3.18: Comparaison par rapport à la charge des Vms dans un environnement homogène.

Selon la figure 3.18, nous remarquons que notre algorithme a mieux équilibré les tâches contrairement à l'approche sans traitement (l'arrivée aléatoire des tâches), une chose qui est logique. Le NDL-LPT est aussi mieux que le Round-Robin qui n'est pas scalable pour les grandes instances mais qui est censé donner des bonnes performances pour les petites instances.

Le makespan représente le temps de terminaison de la dernière tâche et par conséquent la fin de l'équilibrage de tâches. Notre algorithme surpasse le Round-Robin et bien évidement le cas sans traitement vers la fin de l'équilibrage. Nous remarquons cet écart dans la figure 3.19 à partir de 450 tâches. En absence d'équilibrage, comme c'est indiqué dans la figure 3.18, le makespan n'est pas optimisé car la dernière tâche peut se trouver affectée à une machine qui est déjà surchargée. Avec le NDL, les VMs reçoivent des tâches de manière équilibrée et par conséquent le makespan sera réduit.

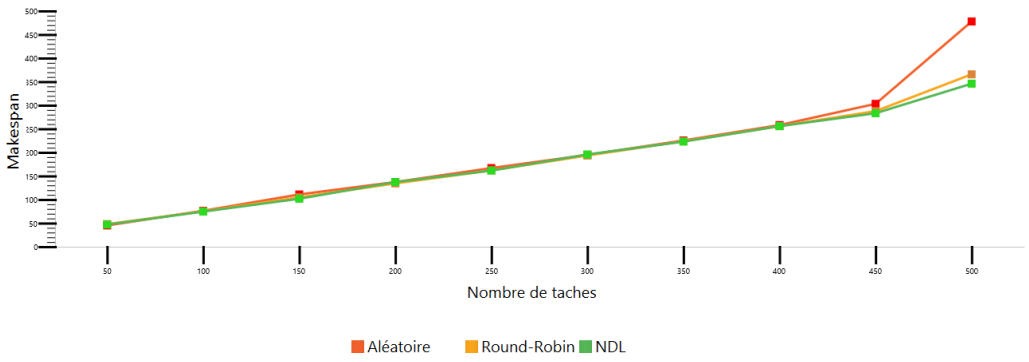

FIG. 3.19: Comparaison par rapport au makespan dans un environnement homogène.

3.6.2 Dans un environnement hétérogène et avec différentes métriques

Dans ce qui suit, nous présentons l'évaluation de notre algorithme dans un environnement hétérogène. Notre configuration contient un nombre variable de Datacenters, de hôtes et de Vms. Les hôtes et les VMs sont distribués sur les datacenters selon le Round Robin. Les VMs sont caractérisées par : un nombre variable de cores (entre 1 et 4), une capacité variable de RAM (entre 1024 et 6144 MB avec un pas de 1024 MB) et une capacité de bande passante égale à de 100 MB/s pour toutes les VMs. Nous avons simulé l'arrivée de différentes tâches d'une longueur variable (exponentielle) et générée d'une manière aléatoire entre 10 000 et 80 000 MI avec des temps d'arrivées aléatoires des tâches entre 0 et 10 secondes.

3.6.2.1 Impact du nombre de tâches sur les performances

Dans ce qui suit, nous allons considérer une configuration contenant 10 datacenters, 50 hôtes et 100 Vms. Le nombre de tâches sera varié entre 100 et 1000 afin de mesurer les performances du NDL avec BF et avec LPT.

— **Temps de réponse moyen**

La figure 3.20 représente le temps de réponse moyen pour exécuter toutes les tâches en fonction du nombre de tâches pour les deux stratégies LPT et BF.

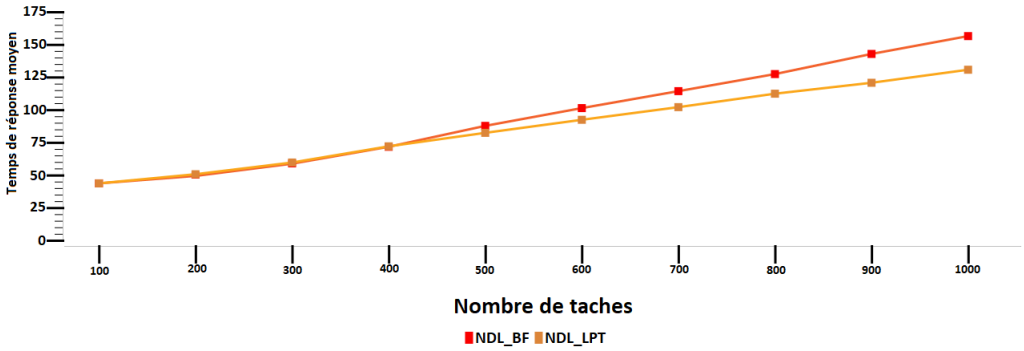

FIG. 3.20: Temps de reponse moyen versus nombre de tâches.

Pour les deux stratégies, le temps de réponse moyen pour exécuter toutes les tâches augmente en fonction du nombre de tâches. Par contre, le LPT surpasse le BF à partir de 600 tâches et ça en raison de la recherche exhaustive du BF afin d'équilibrer les pairs.
Si le nombre de tâches est important, le BF piétine contrairement au LPT qui continue à fonctionner (nous avons remarqué ça à partir de 1100 tâches pour cette configuration).

— **Temps d'inactivité moyen**

La figure 3.21 représente le temps d'inactivité moyen pour exécuter toutes les tâches en fonction du nombre de tâches pour les deux stratégies LPT et BF.

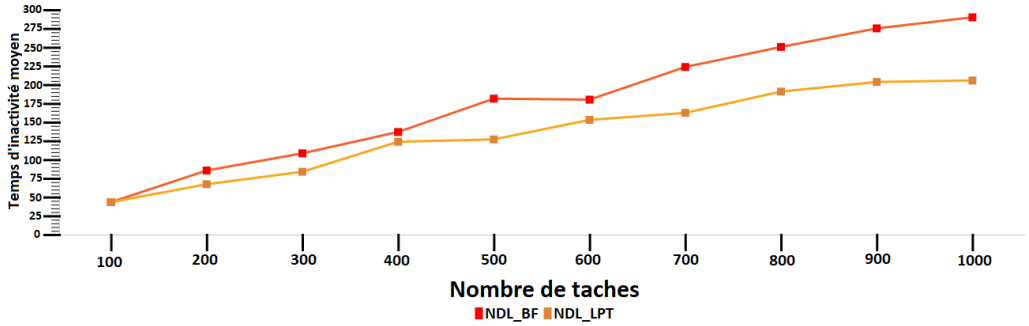

FIG. 3.21: Temps d'inactivité moyen versus nombre de tâches.

Le temps d'inactivité moyen augmente avec l'augmentation du nombre de tâches car lié à la disponibilité des Vms afin de traiter les tâches supplémentaires. Par contre ce temps est plus important pour le BF que le LPT car le BF fait plus de calcul afin de trouver la solution optimale locale contrairement au LPT et par conséquent les Vms attendent plus pour le BF que pour le LPT afin de récupérer des tâches affectées.

— **Makespan**

La figure 3.22 représente le makespan en fonction du nombre de tâches pour les deux stratégies LPT et BF.

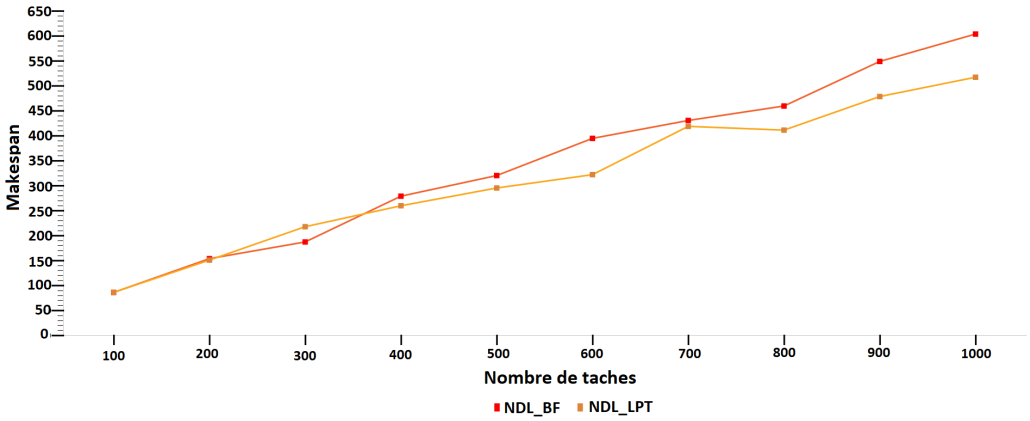

FIG. 3.22: Makespan versus nombre de tâches.

Pour les deux stratégies, le makespan augmente en fonction du nombre de tâches. Par contre, le LPT surpasse le BF à partir de 400 tâches et ça en raison de la recherche exhaustive du BF afin d'équilibrer les pairs. Si le nombre de tâches est réduit, le makespan pour le BF surpasse ou pratiquement comparable avec celui du LPT.

3.6.2.2 Impact du nombre de Vms sur les performances

Dans ce qui suit, nous allons considérer une configuration contenant 5 datacenters et 10 hôtes. Nous fixons le nombre de tâches à 300. Le nombre de VMs sera varié entre 10 et 50 afin de mesurer les performances du NDL avec BF et avec LPT.

— **Temps de réponse moyen**

La figure 3.23 représente le temps de réponse moyen pour exécuter toutes les tâches en fonction du nombre de VMs pour les deux stratégies LPT et BF.

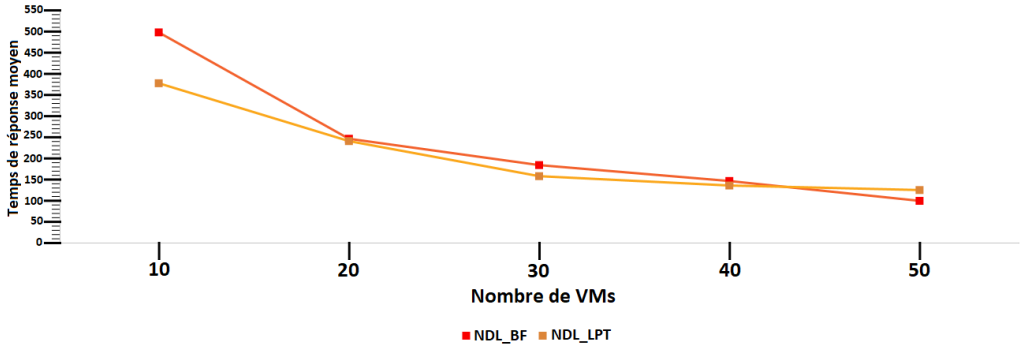

FIG. 3.23: temps de réponse moyen versus nombre de Vms.

En fixant le nombre de tâches et avec l'augmentation du nombre de ressources, les Vms auront moins d'équilibrages à faire et par conséquent le temps de réponse moyen sera réduit.

Selon la figure 3.23, nous remarquons que le LPT surpasse le BF avec moins de ressources (surtout dans le cas avec 10 VMs). Dans ce cas, il y aura plus d'équilibrages à faire pour les deux stratégies mais le cout de la recherche exhaustive pénalise le BF.

— **Temps d'inactivité moyen**

La figure 3.24 représente le temps d'inactivité moyen pour exécuter toutes les tâches en fonction du nombre de VMs pour les deux stratégies LPT et BF.

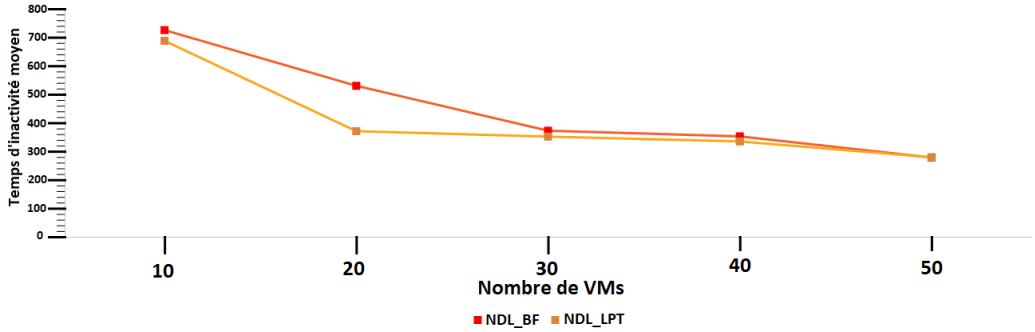

FIG. 3.24: Temps d'inactivité moyen versus nombre de Vms.

Selon la figure 3.24, nous remarquons que plus il y aura des Vms dans le Cloud, moins il y aura d'inactivité pour chaque VM durant l'équilibrage, parce que ces dernières aurons moins de tâches à exécuter, et par conséquent le temps d'attente moyen diminuera.

Les VMs utilisant LPT ont moins d'inactivité par rapport a ceux utilisant BF car ce dernier perd du temps a faire les itérations a chaque équilibrages de charge et donc les VMs peuvent attendre plus de temps pour recevoir des tâches à exécuter.

— **Nombre de migrations**

La figure 3.25 représente le nombre de migrations pour exécuter toutes les tâches en fonction du nombre de VMs pour les deux stratégies LPT et BF.

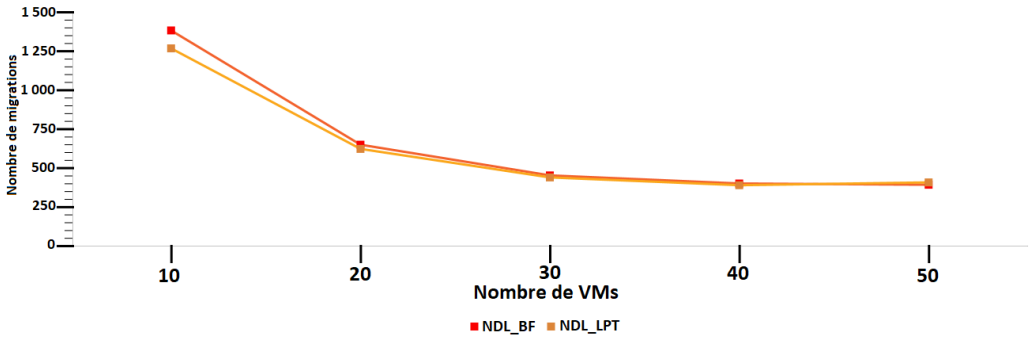

FIG. 3.25: Nombre de migrations versus nombre de Vms.

Selon la figure 3.25, nous remarquons que plus il y aura des Vms dans le Cloud, moins il y aura de migrations à faire.

En effet, plus de ressources implique moins de déséquilibre et donc moins de migrations. La migration dépend sensiblement de la distribution uniforme ou biaisée (effet sporadique/aléatoire) de l'arrivée des tâches sur les VMs du réseau. Si c'est plus ou moins équilibré en termes d'arrivées, on aura moins de migrations à faire, mais si c'est déséquilibré, on aura plus de migrations à faire. Aussi, nous remarquons que le BF fait plus de migration par rapport au LPT pour un nombre de VMs inférieur à 20, une chose qui est liée à la stratégie du BF. La différence entre les deux est a peine visible et non significative, lorsque le nombre de VMs est élevé.

— **Makespan**

La figure 3.26 représente le makespan en fonction du nombre de VMs pour les deux stratégies LPT et BF.

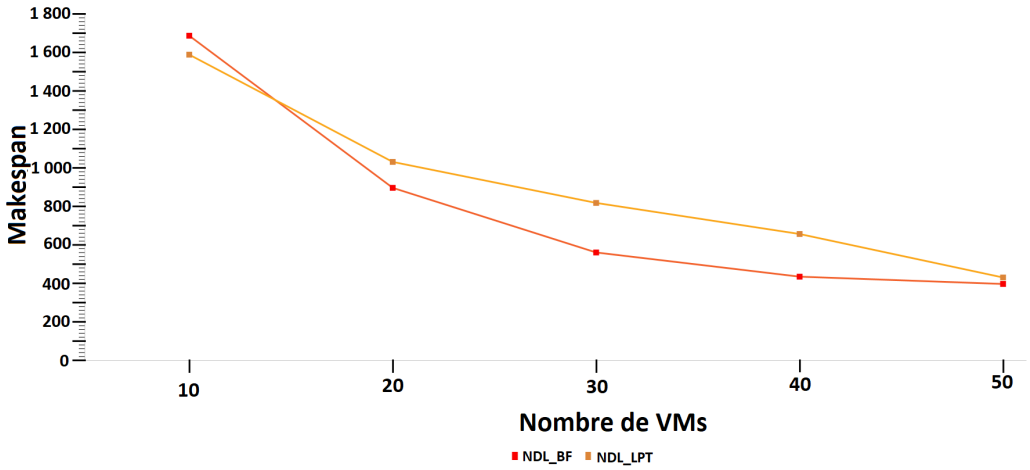

FIG. 3.26: Makespan versus nombre de noeuds.

En fixant le nombre de tâches et avec l'augmentation du nombre de ressources, les Vms auront moins d'équilibrages à faire et par conséquent le makespan sera réduit.

Selon la figure 3.26, nous remarquons que le LPT surpasse le BF avec moins de ressources (surtout dans le cas avec 10 VMs). Dans ce cas, il y aura plus d'équilibrages à faire pour les deux stratégies mais le cout de la recherche exhaustive pénalise le BF. Le BF surpasse le LPT pour un nombre de VMs entre 20 et 40. Dans ce cas, le BF a mieux équilibré les tâches et par conséquent le makespane était meilleur. LPT et BF ont des performances similaires avec 50 Vms car moins d'équilibrages à faire pour les deux.

Avec 50 Vms on observe presque le même temps car il y aura beaucoup moins d'équilibrage à faire.

— **Écart par rapport à la solution idéale**

Dans ce qui suit, nous allons évaluer l'impact de la stratégie utilisée pour l'équilibrage sur l'écart par rapport à la solution idéale (charges équitables sur l'ensemble des VMs). Pour cela, nous avons opté pour l'écart-type, ce dernier sert à mesurer la dispersion, ou l'étalement, d'un ensemble de valeurs autour de leur moyenne. Plus l'écart-type est faible, plus les valeurs sont bien étalées autour de leur moyenne.
Si $\bar{x}$ est la moyenne des charges de toutes les VMs et si la stratégie utilisée donne à la fin de l'équilibrage la solution t_i affectée à VM_i. L'écart-type pour l'ensemble des charges pour cette stratégie est calculé comme suit : $\sqrt{\frac{\sum_{i=1}^{N}(t_i-\bar{x})^2}{N}}$, (N : nombre de Vms).
La meilleure stratégie est celle qui possède la valeur minimale.
La figure 3.27 représente l'écart des deux stratégies de la solution idéale (0 dans le cas des tâches divisibles) en fonction du nombre de VMs.

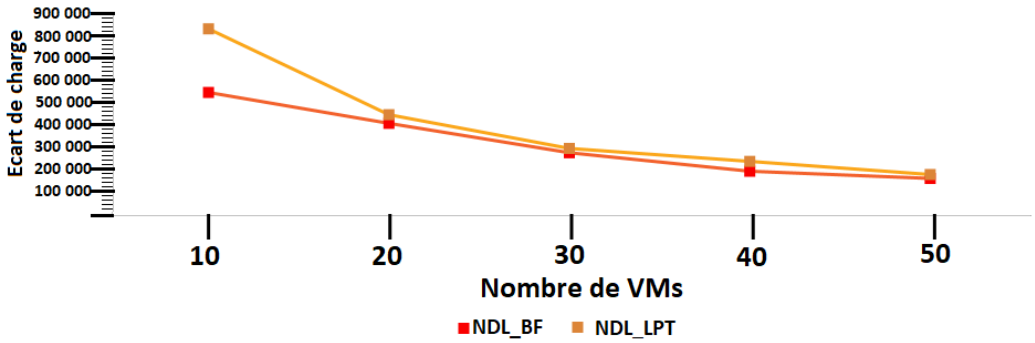

FIG. 3.27: Écart versus nombre de Vms.

Dans l'ensemble et comme indiqué dans la figure 3.27, le BF à mieux équilibrer la charge par rapport au LPT (surtout pour un nombre de VMs inférieur à 20). La raison est claire, le BF fourni en local, la solution optimale contrairement au LPT qui fourni (toujours en local), la solution approchée. Avec moins de VMs, le BF a un contexte favorable pour montrer ses performances (plus

d'équilibrages à faire). Avec l'augmentation du nombre de VMs, le LPT arrive à concurrencer le BF car visiblement moins d'équilibrage à faire et par conséquent des performances similaires.

Pour conclure, nous pouvons affirmer que le BF est nettement meilleur par rapport au LPT en termes d'équilibrages de charges. Le BF par contre n'est pas scalable à cause de son cout en termes de complexité pour fournir la solution optimale (blocage avec 1100 tâches et 100 VMs). Le LPT reste le choix à prendre en cas de passage à l'échelle et a donné des bonnes performances par rapport à la plus part des métriques étudiées.

3.7 Conclusion

Ce chapitre a été consacré à la présentation de notre contribution dans le cadre ce projet de fin d'études. Il s'agit d'implémenter et d'évaluer une approche d'équilibrage de charges appliquée au Cloud computing. Les différentes simulations ont été réalisées à l'aide de l'outil CloudSim Plus, un simulateur performant dans le domaine du Cloud. Notre approche est basée sur un algorithme distribué utilisant deux stratégies. La première est basée sur une méthode approchée qui est le LPT (Largest Processing Time first) fournissant une solution approchée en local, la seconde est basée sur une recherche exhaustive (brute force) fournissant une solution optimale (toujours en local).

Conclusion générale et perspectives

Au cours de ce mémoire, nous nous sommes penchés sur le problème d'équilibrage de charges dans le Cloud Computing, l'informatique du futur, tel qu'il est décrit par « GAFAM », les géants du Web : Google, Apple, Facebook, Amazon et Microsoft.

Nous avons commencé par introduire les définitions de base nécessaires à la compréhension du Cloud, son architecture et ses différents types (privée, public, hybride) et services (IaaS, PaaS, SaaS). Par la suite, nous avons cerné le problème d'équilibrage de charges et ses divers politiques et mécanismes.

Plus tard nous avons présenté le simulateur CloudSim Plus, dans lequel nous avons fait un jeu de test de notre algorithme distribué suivant divers métriques : le makespan, le temps de réponse moyen, le temps d'inactivité moyen et le nombre de migrations. L'objectif visé est de faire une comparaison entre l'approche LPT (Largest Processing Time first) et l'approche BF (Brute Force), les deux stratégies utilisées dans les transactions pair-wise.

Suite aux différentes simulations réalisées dans le cadre de ce travail, nous avons conclu que le BF est nettement meilleur par rapport au LPT en termes d'équilibrages de charges. Son écart par rapport à la solution idéale reste meilleur que celui de LTP. La recherche exhaustive par contre n'est pas scalable à cause de son cout en termes de temps d'exécution afin de fournir l'équilibrage optimale en local. Le LPT supporte le passage à l'échelle fournissant en local une solution proche de l'optimal et a donné de bonnes performances par rapport à la plus part des métriques évaluées.

Suite à la réalisation de notre étude, il serait intéressant de se comparer avec des méthodes approchées (les métaheuristiques) comme les algorithmes génétiques, l'optimisation par essaims particulaires, cuckoo search ou flower pollination algorithm afin de se comparer avec les résultats obtenus par la stratégie LPT du NDL.

Finalement, nous espérons que les objectifs fixés au départ ont été en grande partie réalisés et que les résultats obtenus trouveront un bon écho.

Bibliographie

[1] A Thorough Introduction to Distributed Systems. Disponible Sur :https ://www.freecodecamp.org/news/a-thorough-introduction-to-distributed-systems-3b91562c9bc/. Consulté le :10 avril 2020.

[2] What Is A Computer Network Cluster.Disponible sur https ://www.eukhost.com/blog/webhosting/what-is-a-computer-network-cluster. Consulté 27 fevrier 2020.

[3] What is the difference between Cluster, Grid and Cloud ?.Disponible sur : https ://www.stratoscale.com/blog/cloud/geek-talk-what-is-the-difference-between-cluster-grid-and-cloud. Consulté le 28 fevrier 2020 .

[4] Someone else's computer : The prehistory of cloud computing. https ://spectrum.ieee.org/tech-history/silicon-revolution/someone-elses-computer-the-prehistory-of-cloud-computing. Consulté le : 10 avril 2020.

[5] CLOUDCOMPUTING ARCHITECTURE : LAYERS OF CLOUDPYRAMID. Disponible sur :https ://medium.com/@FedakV/cloud-computing-architecture-layers-of-cloud-pyramid-b2e70f877c91.
Consulté le : 19 mars 2020.

[6] Cloud Computing – Définition, Avantages et Exemples d'utilisation. Disponible sur : https ://www.lebigdata.fr/definition-cloud-computing Consulté le : 20 mars 2020,.

[7] Qu'est-ce qu'un Cloud public ?. Disponible Sur https ://www.vmware.com/fr/topics/glossary/content/public-cloud.html. Consulté le : 1 avril 2020.

[8] Qu'est-ce qu'un Cloud privé ?. Disponible Sur https ://www.vmware.com/fr/topics/glossary/content/private-cloud.html. Consulté le : 1 avril 2020.

[9] Qu'est-ce qu'un Cloud hybride ?. Disponible Sur https ://www.vmware.com/fr/topics/glossary/content/hybrid-cloud.html. Consulté le : 1 avril 2020.

[10] Virtualization in Cloud computing. Disponible Sur :https ://www.educba.com/virtualization-in-cloud-computing/. Consulté le : 26 avril 2020.

[11] Que se cache derrière le cloud computing ?Disponible Sur :https ://www.ionos.fr/digitalguide/serveur/know-how/cloud-computing. Consulté le :5 avril 2020.

[12] Yagoubi, Bellabas. Modèle d'équilibrage de charge pour les grilles de calcul. Revue africaine de la recherche en informatique et mathématiques appliquées (ARIMA). Vol. 7. pp. 1-19. 2007.

[13] Yagoubi, Bellabas. Modèle Arborescent pour l'équilibrage de charge dans les Grilles de Calcul. CARI 2006. 8 ème colloque africain sur la recherche en Informatique. 6-9 novembre 2006. Cotonou. Bénin.

[14] Wautelet, Frédéric. Régulation de charge dans les systèmes distribués : architecture et algorithmes. Mini-Workshop Systèmes Coopératifs. Matière Approfondie. Disponible sur https ://staff.info.unamur.be/ven/cisma/files/2002- wautelet.pdf. Consulté le : 4 mai 2020.

[15] Anne-Marie Charles. Problèmes d'ordonnancement avec ressources. Cours présenté à l'Université Paris-Dauphine. Disponible sur http ://ressources.aunege.fr/nuxeo/site/esupversions/2b1c56b6-109d488a-94a3- 3ea525f8beef/modaiddec/cours/l11/l11.pdf. Consulté le : 5 mai 2020.

[16] Martha Rosa CASTAÑEDA RETIZ. Étude quantitative des mécanismes d'équilibrage de charge dans les systèmes de programmation pour le calcul parallèle. Thèse de Doctorat en informatique. 1999. Institut polytechnique de Grenoble.

[17] Régulation de charge dans les systèmes distribués : architecture et algorithmes . Disponible Sur https ://staff.info.unamur.be/ven/CISma/FILES/2002-wautelet.pdf. Consulté le : 4 mai 2020.

[18] Load-Balancing Algorithms. Disponible Sur https ://community.cengage.com/Infosec2/f/20/t/2353. Consulté le : 6 mai 2020.

[19] Yagoubi, Bellabas. Equilibrage de charge dans les grilles de calcul. Thèse de Doctorat d'état en informatique. 2007. Université d'Oran.

[20] Qu'est-ce que la technologie Java et pourquoi en ai-je besoin ?. Disponible Sur https ://www.java.com/fr/download/faq/whatis java.xml. Consulté le : 11 avril 2020.

[21] JavaFX aide à créer l'interface d'une application Java. Disponible Sur https ://www.scriptol.fr/programmation/javafx.php. Consulté le : 11 avril 2020.

[22] Organisez et packagez une application Java avec Apache Maven. Disponible Sur https ://openclassrooms.com/fr/courses/4503526-organisez-et-packagez-une-application-java-avec-apache-maven. Consulté le : 11 avril 2020.

[23] Plateforme IntelliJ. Disponible Sur https ://www.jetbrains.com/fr-fr/opensource/idea/. Consulté le : 11 avril 2020.

[24] Qu'est-ce que SceneBuilder ?. Disponible Sur https ://java.developpez.com/faq/javafx ?page=SceneBuilder. Consulté le : 11 avril 2020.

[25] Silva Filho, Manoel C., et al. CloudSim plus : a cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). IEEE, 2017. Lisbon, Portugal, May 8-12, 2017. p. 400-406.

[26] VisFX. Disponible sur https ://github.com/arocketman/VisFX. Consulté le : 15 mai 2020.

Résumé :

La quantité de données à stocker et à traiter dans le Cloud augmente de jour en jour et d'une façon très rapide, ce qui nécessite un mécanisme qui permet d'équilibrer la charge de stockage. Dans le cadre de ce travail, nous avons implémenté et évalué un nouvel algorithme distribué pour les charges indivisibles appliqué au Cloud Computing. L'algorithme est inspiré du problème de sacs à dos multiple. Le simulateur CloudSim Plus a été utilisé dans le cadre de ce travail et notre approche a été évaluée à l'aide de deux stratégies ; à savoir : LPT (Largest Processing Time first) et BF (Brute Force). Les résultats obtenus montrent la supériorité de LPT par rapport à BF en termes de différentes métriques et aussi pour le passage à l'échelle.
Mots clés : Système distribué, Cloud computing, Équilibrage de charge, LPT, BF.

Abstract :

The amount of data to store and process in the Cloud is increasing day by day and very quickly, which requires a mechanism to balance the storage load. As part of this work, we implemented and evaluated a new self-stabilizing algorithm for indivisible loads applied to Cloud Computing. The algorithm is inspired by the multiple knapsack problem. The CloudSim Plus simulator was used in this work and our approach was evaluated using two strategies ; namely : LPT (Largest Processing Time first) and BF (Brute Force). The results obtained show the superiority of LPT compared to BF in terms of different metrics and also for scalability.
Keywords : Distributed system, Cloud computing, Load balancing, LPT, BF.

ملخص :

يزداد حجم البيانات المخزنة والمعالجة فيزداد حجم البيانات التي يتم تخزينها ومعالجتها في السحاب يومًا بعد يوم وبسرعة كبيرة ، الأمر الذي يتطلب آلية لموازنة حمل التخزين. كجزء من هذا العمل ، قمنا بتطبيق وتقييم خوارزمية جديدة لتحقيق الاستقرار الذاتي للأحمال غير القابلة للتجزئة المطبقة على الحوسبة السحابية. الخوارزمية مستوحاة من مشكلة الحقائب المتعددة. تم استخدام محاكي CloudSim Plus في هذا العمل وتم تقييم نهجنا باستخدام استراتيجيتين ؛ وهي: LPT و BF . أظهرت النتائج التي تم الحصول عليها تفوق LPT مقارنة مع BF من حيث المقاييس المختلفة وكذلك للتوسع.
الكلمات المفتاحية : النظام الموزع ، الحوسبة السحابية ، موازنة التحميل، LPT ، BF .