ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE AVEC MÉMOIRE EN GÉNIE
CONCENTRATION : RÉSEAUX DE TÉLÉCOMMUNICATIONS
M. Sc. A.

PAR
Hicham ABDELFATTAH

ANALYSE DE PERFORMANCE PAR PLANS EXPÉRIMENTAUX D'INTERGICIELS INFONUAGIQUES; LE CAS D'OPENSTACK

MONTRÉAL, LE 4 MAI 2016

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Mohamed Cheriet, directeur de mémoire
Département de génie de la production automatisée à l'École de Technologie Supérieure

M. Abdelouahed Gherbi, président du jury
département de génie logiciel et des TI à l'École de Technologie Supérieure

M. Kim Khoa Nguyen, membre du jury
département de génie électrique à l'École de Technologie Supérieure

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

LE 26 AVRIL 2016

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ANALYSE DE PERFORMANCE PAR PLANS EXPÉRIMENTAUX D'INTERGICIELS INFONUAGIQUES; LE CAS D'OPENSTACK

Hicham ABDELFATTAH

RÉSUMÉ

Les récents progrès dans la technologie de cloud computing ont fourni aux entreprises une occasion d'accroître leur profit en se concentrant davantage sur leurs business sans se soucier de l'infrastructure matérielle sous-jacente. D'autre part, les opérateurs de cloud sont confrontés à de nouveaux défis dans la gestion et le maintien de l'installation complexe comprenant une variété de calcul informatisé (computing), du réseau, de l'énergie et de l'infrastructure de refroidissement, ainsi que les plates-formes de logiciels de cloud.

Pour améliorer leur efficacité, les opérateurs de cloud requièrent des informations pertinentes sur leur infrastructure, en particulier le comportement de la plate-forme de gestion de cloud, que l'on appelle le Cloud Middleware. Une analyse globale de la performance du Cloud Middleware permettra d'identifier ses forces et ses limites. Cependant, cette analyse est très difficile en raison du grand nombre de composants du Cloud Middleware, et leurs connexions en silos. Dans cette thèse, nous présentons une approche de profilage basée sur les plans d'expériences pour répondre à ce besoin. Nous évaluons la performance de OpenStack comme un Cloud Middleware basé sur différents scénarios pour déterminer le comportement des composants OpenStack et d'évaluer sa performance.

Nos expériences ont montré que le facteur le plus influence sur la performance du Cloud est le nombre d'instances créées et en cours d'exécution. Également en changeant les configurations de OpenStack, ce dernier peut supporter une grande quantité de charge allant jusqu'à 800 instances. Nous concluons que la performance d'OpenStack est contrôlée par deux paramètres qui sont corrélés ; le nombre de nœuds de calcul et le nombre de travailleurs à travers les Nova Controller et Neutron serveurs. Par ailleurs, la ressource consommée par OpenStack en termes d'unité centrale de traitement (CPU) augmente avec le nombre d'instances. La version actuelle de OpenStack (IceHouse) ne peut pas supporter plus de 800 instances et le temps nécessaire pour la création d'instances augmente de façon linéaire avec le nombre d'instances.

Mots clés: Virtualisation, Nuage Informatique, Interlogiciel, profilage, OpenStack, consommation des ressources, Plan d'expériences.

OPENSTACK ASSESSMENT AS A CLOUD MIDDLEWARE USING DESIGN OF EXPERIMENT APPROACH. APPLICATION : OPENSTACK

Hicham ABDELFATTAH

ABSTRACT

Recent advances in cloud computing technology have provided companies with an opportunity to increase their profit by focusing more on their business with no worries about underlying hardware infrastructure. On the other hand, cloud operators are facing new challenges in managing and maintaining complex facility comprising a variety of computing, network, energy, and cooling infrastructure, as well as cloud software platforms.

To improve their efficiency, cloud operators require relevant information about their infrastructure, in particular the behaviour of the cloud management platform, so called the cloud middleware. A holistic analysis of cloud middleware performance will help pinpoint on its strengths and limitations. However, such analysis is very challenging due to the large number of cloud middleware components, and their siloed connections. In this thesis, we present a profiling approach based on design of experiments to meet this need. We evaluate the performance of OpenStack as a cloud middleware based on different scenarios to determine the behavior of OpenStack components and evaluate its performance.

Our experiments showed that the most influencing factor on the performance of the cloud is the number of instances created and running. Also by changing the OpenStack configurations, it can support a larger amount of workload up to 800 instances. We conclude that the performance of OpenStack is controlled by two metrics that are correlated ; the number of compute nodes and the number of workers across Nova Controller and Neutron servers. Furthermore, resource consumed by OpenStack in terms of CPU increases along with the number of instances. The current version of OpenStack (IceHouse) can not support more than 800 cases and the time required for creating instances increases linearly with the number of instances.

Keywords: Virtualization, Cloud Computing, Cloud Middleware, Profiling, OpenStack, Resource Consumption, Design of Experiments.

REMERCIEMENTS

Au terme de ce mémoire, je présente mes remerciements les plus sincères à mon encadrant du projet, Monsieur Mohammed Cheriet, Professeur titulaire à l'École de technologie supérieure et Directeur du laboratoire Synchromedia, pour son soutien, ses encouragements et ses orientations précieuses, malgré les occupations et les responsabilités qu'il assume, durant ma période du projet. Ainsi, je souhaite que ce travail soit le modeste témoignage de ma haute considération et mon profond respect pour mon honorable professeur.

Je voudrai aussi témoigner mes remerciements, à Monsieur Kim Nguyen, Professeur assistant et associé de recherche, au laboratoire Synchromedia, pour son appui et son soutien tout au long de ces deux années, pour ses efforts indéniables afin de nous procurer un meilleur apprentissage. Aussi, je tiens à remercier vivement les membres du jury, pour leur de participation à l'évaluation de ce travail.

Mes remerciements s'adressent également aux membres du laboratoire Synchromedia, à mes amis, mes frères et ma sœur qui ont été toujours présents pour moi et qui m'ont toujours soutenu et encouragé. Enfin, un grand merci à l'ensemble des personnes qui ont participé de près ou de loin à l'élaboration de ce travail.

À vous tous, je dis merci,

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

IDC	International Data Corporation
OPEX	Operational Expenditures
CAPEX	Capital Expenditures
ERP	Enterprise Resource Planning
SOA	Service-Oriented Architecture
EC2	Elastic Compute Cloud
S3	Amazon Simple Storage Service
CTPD	Integrated Tracing Profiling and Debugging
LTTng	Linux Trace Toolkit Next Generation
NIST	National Institute of Standards and Technology
IaaS	Infrastructure as a Service
SaaS	Software as a Service
PaaS	Platform as a Service
NFS	Network File System
SGE	Sun Grid Engine
OS	Operating System
NIC	network Interface Controller
IP	Internet Protocol
MAC	Media Access Control
SSH	Secure Shell
VM	Virtual Machine
KPI	Key Performance Indicator
CCA	Canonical Correlation Analysis

I/O	In/Out
CPU	Central Processing Unit
RAM	Random Access Memory
SLA	Service-Level Agreement
PET	Program Execution Time
DPI	deep packet isnspection
TAP	Test Access Point
RPC	Remote Procedure Call
API	Application Program Interface
VIF	Virtual Network Interface
DHCP	Dynamic Host Configuration Protocol
SS7	Signaling System 7
OSS	Office of Strategic Services

CHAPITRE 1

INTRODUCTION

1.1 Contexte général

L'ultime question que les grandes entreprises informatiques se posent tant au niveau des fournisseurs de services qu'au niveau des opérateurs est : comment peut-on garantir des services toujours disponibles, faciles à manipuler pour les utilisateurs et avec moins d'implication de ces derniers, accessibles partout et en même temps rentables ! ? Ainsi, pour offrir un tel service avec une telle exigence, cela nécessite autant de travail conceptuel que de ressources informatiques, unité de calcul et de mémoire, pour acquérir un tel besoin.

Au début, afin de procurer et garantir un tel service, nous avons besoin des stations dédiées juste pour l'infrastructure qui doivent être en même temps bien structurées, complexes, et interconnectées pour héberger ces services. De plus, en prenant en considération la consommation d'énergie électrique, la climatisation, la gestion et la maintenance de ces immenses matériels que ces entreprises payent par jour, ces facteurs impactent directement et négativement sur le profit de l'entreprise ou ce que nous appelons le retour sur investissement (RSI) et augmentent le coût total de possession (CTP). D'après une étude qui a été faite par International Data Corporation (IDC) (boursas et al., 2008) nous estimons que les serveurs en entreprise utilisent jusqu'à 15% de la capacité totale de leurs ressources matérielles. Nous tirons de cette étude qu'il y a un grand écart entre l'achet et les retombés de ces serveurs.

En contrepartie, le concept de la virtualisation joue un rôle primordial pour remédier et restreindre cet écart. Cette technologie a un impact remarquable et pertinent dans le domaine de la technologie de l'information et de la communication. Selon Hervé Renault (Directeur Technique Europe du Sud pour VMware), la virtualisation ne s'arrête pas aux serveurs, mais touche également les applications critiques telles que les ERP, les bases de données ou les messageries. Il expose les bénéfices de cette évolution majeure à commencer par la réduction significative

des coûts. (boursas et al., 2010). Avec la virtualisation , nous économisons jusqu'à 60 % sur les coûts d'investissement informatique, 30 % de réduction des coûts opérationnels et 80 % de la facture énergétique.

Toujours dans la même vague d'innovation, d'autres technologies ont émergé : en l'occurrence le nuage informatique (Cloud Computing) qui est un concept d'entreprise, englobant à la fois la virtualisation, la grille informatique, la technologie SOA et bien évidemment d'autres aspects informatiques. D'après la firme Brocade (Brocade,2012), 60 % des entreprises envisagent une migration vers le nuage informatique depuis l'année 2012. Ce concept permet aux entreprises de se focaliser au cœur du business, avec l'apparition des solutions commerciales, comme EC2 et S3 d'Amazon, Microsoft Azure (Microsoft, 2012). Actuellement, le system du cloud office représente 8% de la totalité du marché des offices et atteindra 33% dans l'horizon de l'année 2017(gartner, 2013). Vu les avantages que le nuage informatique nous offre, de nombreuses entreprises décident d'étendre leurs infrastructures vers le nuage informatique.

Cependant, déployer de nouvelles solutions ou migrer des solutions existantes n'est pas une tâche facile ou négligeable qui se fait en quelques étapes. Par contre, cela est un processus assez complexe vu le grand nombre d'opérations prises en considération : comme l'utilisation de plusieurs services à la fois distribués et répartis dans le réseau, et nécessitant des bases de données d'un middleware et d'autres composants à configurer.

Pour garantir le bon fonctionnement et déroulement du cloud avec moins de tolérance des erreurs, nous avons recours au monitoring qui est un élément essentiel pour un tel environnement comme le Cloud. Le monitoring est un processus ayant pour but de faire le suivi et la gestion du flux de travail qui circule dans le cloud : allant du bas niveau soit l'infrastructure, arrivant au niveau haut soit l'application et en passant par le middleware assure la communication entre les deux extrémités. De plus, le monitoring aide également à évaluer la performance de l'ensemble du cloud à un niveau modulaire.

L'émergence du concept du cloud computing chaperonne par plusieurs travaux de recherche, dont deux ont collaboré avec ce mémoire. Le projet EcoloTIC de la compagnie de télécommunications Ericson ayant pour mission de développer une solution intégrale offrant entre autres une architecture de cloud computing pour les différentes technologies avec une basse consommation d'énergie et à faible empreinte de carbone. En effet, cela est réalisé par l'analyse, le contrôle et la connaissance précise de la consommation des ressources afin d'optimiser et d'utiliser ces derniers d'une manière efficace, d'une part, et d'envoyer ces analyses de la performance au gestionnaire du cloud qui a pour rôle de manœuvrer le déroulement du système entier , d autre part.

Aussi, le projet CTPD(Cloud Tracing, Profiling and Debugging) de l'équipe de recherche sur le LTTng (DORSAL,2013) adapte son outil de traçage du système d'exploitation Linux, pour tracer les différentes communications entrant/sortant du système d'exploitation, des applications et des différentes couches de la virtualisation, que la compagnie Ericson soutient dans ce projet. En développant des méthodologies appropriées dans les environnements du cloud computing, ces dernières donneront un plus au projet EcoloTIC par la vérification des messages circulant dans le système et par la détection des messages d'erreurs.

Ceci donne une idée mature sur la vision et la stratégie des grandes compagnies en investissant sur ces nouvelles approches. Avec le profilage du cloud middleware, en déterminant ces points forts avec ces limitations, notre projet de recherche jouera un rôle assez important dans les deux projets.

1.2 Objectif

En effet, le Cloud Computing englobe et utilise plusieurs techniques et technologies à la fois. En outre, le Cloud Computing se base sur plusieurs aspects dont le modèle de service qui lui aussi se décompose en trois modèles, respectivement infrastructure en tant que Service, Plate-forme en tant que Service et Logiciel en tant que Service. Par conséquent, avant de nous

plonger dans la recherche scientifique, nous sommes censés fixer notre travail de recherche. De ce fait, il est très important de mettre en évidence et de souligner les objectifs de cette recherche en élaborant une stratégie de méthodologie du travail afin de bien profiter de cette expérience académique.

Comme dans toute technologie, le Cloud Computing s'appuie sur des modèles architecturaux. Le modèle de service qui nous intéresse le plus, est le modèle infrastructure en tant que service. Nous pouvons dire que ce service est la base ou la première couche inférieure dans le Cloud Computing. Notre recherche sera fondée plus précisément sur le Cloud Middleware qui est, entre autre, une couche sous-jacente permettant de réaliser cette infrastructure du Cloud.

L'objectif principal de cette présente recherche est de développer une méthodologie de profilage ayant pour but d'évaluer, d'estimer et de calculer la performance du Cloud Middleware d'une manière précise et d'établir une étude pointue de l'impact du Cloud Middleware sur les ressources matérielles utilisées.

En effet, ce genre de profilage du Middleware permettra la supervision et la prédiction de sa performance sur des différentes infrastructures. Il nous aidera à mettre le doigt sur la(s) composante(s) du middleware cloud ayant un impact direct sur les ressources physiques utilisées. Ceci a pour but d'améliorer l'efficacité du Cloud. Car nous avons deux autres services qui s'ajoutent au-dessus de cette couche. Par conséquent, la performance de cette dernière joue un rôle primordial sur la performance de la totalité du cloud computing.

Le profilage dans le Cloud Middleware est généralement une approche que nous utilisons afin de nous aider à répondre à la question : pourquoi de certaines conditions nous obtenons un tel résultat, cela permet de donner une idée sur l'origine du problème. Aussi, cela nous aide à connaitre et à détecter la limite de saturation ou la dégradation de la performance du système ciblé.

Ceci dit, pour arriver à profiler, il faut tout d'abord modéliser le Cloud Middleware en mettant le point sur les facteurs et les composants du système. Cela est une étape très importante dans le processus du profilage. Ensuite, il faut adopter une méthode pour évaluer les performances du Cloud Middleware d'une manière précise et efficace et surtout moins couteuse en terme de temps d'évaluation.

L'objectif principal est traduit en trois sous-objectifs :
Premièrement, choisir le Middleware souhaité et basé sur des études antérieures. Deuxièmement, extraire les éléments qui entrent en jeux sur la performance du cloud middleware choisi, ceci sera basé sur les plans d'expérience. Troisièmement, nous illustreront un bilan qui montrera l'influence du cloud middleware sur les ressources physiques du serveur, à savoir CPU et la mémoire RAM. Cela mettra le point sur l'impact du Cloud middleware et sur la performance de ces derniers.

1.3 Problématique

Très souvent, nous lions la mauvaise performance au manque des ressources comme (bande passant, stockage, mémoire vive, le cycle de CPU, réseau et autre). Mais, la non satisfaction de la performance est le résidu de plusieurs facteurs.

Nous citons la nature de l'environnement distribué qui peut impacter directement et négativement sur la performance du système entier par la défaillance de l'un de ses composants.

De plus, ces composants sont eux aussi encapsulés dans différentes couches hétérogènes, à savoir matériel physique, type de virtualisation, type de liaison réseau ou système de fichiers utilisé et autres. De ce fait, l'influence de cette dernière est remarquable sur la performance de ces types d'environnement. Tout ceci entraîne une mauvaise distribution des processus sur l'ensemble des ressources disponibles.

Étant donné un Cloud middleware, nous n'avons pas une connaissance préalable et pratique sur le comportement du système sous une chaine de production. En outre, acquérir un apprentissage du fonctionnement du système, nous pousse à ramener une étude qui nous permettra de profiler le cloud middleware. De ce fait, nous répondons aux questions suivantes :

1 - Définir une liste appropriée des paramètres de performances : pour toutes les opérations/-requêtes exécutées sur le Cloud Middleware, il est primordiale de définir précisément et suffisamment l'ensemble des paramètres de la performance du Cloud middleware qui entre en jeux. Le processus du profilage est basé sur cette définition qui caractérise le système étudié. Alors, quelles sont les métriques de performance nécessaires à prendre en considération qui décrivent un Cloud middleware ?

2 - Collecter des données à moindre coût : afin d'obtenir des résultats précis, l'impact des méthodes utilisées pour la collecte des données doivent être négligeable en terme d'influence sur les paramètres de la performance ciblée. Alors, quel est l'outil le plus adéquat qui garantit que les valeurs extractives sont les plus objectives possibles pour la collecte des données de la performance ?

3 - Modéliser la performance du Cloud Middleware : dans le cas de perturbation ou de dégradation des performances des systèmes souvent, nous les joignons à des facteurs qui ne sont pas réels ou qui ne sont pas liés directement aux causes du problème. Ceci revient au manque de connaissance du système ou à la manière de conception de l'étape d'analyse. Alors, de quelle manière pouvons nous extraire les facteurs ou les composants qui impactent directement sur la performance du Cloud Middleware ?

4 - Calculer l'impact des ressources physiques sur la performance du Cloud Middleware : notons ici qu'il aura toujours une couche physique qui gérera le Cloud Middleware que lui-même manipulera les machines virtuelles créées par ce dernier. Ainsi, le flux de communication est très fréquent entre le Middleware et les machines virtuelles. Alors, com-

ment pouvons-nous préciser l'influence de la couche physique et le niveau de saturation de cette dernière sur la performance du Cloud Middleware ?

5 - Évaluer le modèle de performance dans un banc d'essai réel : les expériences des modèles de virtualisation ont prouvé que c'est une approche très économique, d'où très rentable en termes de coût aux entreprises. Mais, dans ce cadre-ci, le Cloud midleware basé sur le concept de virtualisation peut-il être lui aussi fiable et rentable pour les entreprises ?

D'autres questions peuvent être soulevées dans la problématique du profilage dans le Cloud Middleware : notamment, la comparaison de performance et l'impact sur le Cloud Middleware entre les deux types de virtualisation soit le type 1 et le type 2 comme XEN et KVM respectivement. Mais, vu la limite du temps de la maitrise, nous nous limiterons aux questions présentées ci-dessus.

1.4 Plan du mémoire

Notre mémoire s'articule autour de cinq chapitres comme le montre l'organigramme dans la synoptique Fig.1.1.

Chapitre 1 est une introduction générale qui met en évidence le contexte de travail du mémoire, l'objectif de notre recherche et les problèmes reliés à ce dernier.

Chapitre 2 est une revue littérature qui se compose de quatre sections. Nous commencerons par une introduction. Ensuite, la deuxième section sera consacrée au Cloud computing, nous présenterons les modèles de services et du déploiement aussi, nous ferons un survol sur les différentes architectures du Cloud Middleware existantes ainsi qu'une critique des solutions de ces applications déjà présentes. Puis, la troisième section portera sur le monitoring en général et le monitoring dans le Cloud. Entre autre, nous présenterons les travaux connexes de ce sujet de recherche. Nous clôturerons ce chapitre par une conclusion.

Chapitre 3 se focalisera sur la méthodologie de travail optée pour réaliser ce mémoire. Dans cette partie, nous illustrons les hypothèses et la conception du modèle proposé, nous détaillons les différentes études comparatives basées sur des critères bien structurés qui feront entre autres les réponses aux différentes questions et défis présentés dans le premier chapitre de la partie problématique. Ces études seront utilisées dans la réalisation de ce projet de recherche.

Chapitre 4 illustre un cas d'étude concret du profilage d'un Cloud Middleware, et discute les expérimentations effectuées pour valider notre modèle proposé. Enfin, la conclusion de ce travail récapitule le bilan de ce qui est réalisé et les recommandations pour les futurs travaux tant pour les développeurs que pour les utilisateurs du nuage.

Le dernier chapitre sera dédié à synthétiser le travail réalisé, et à présenter des idées pour des travaux futurs.

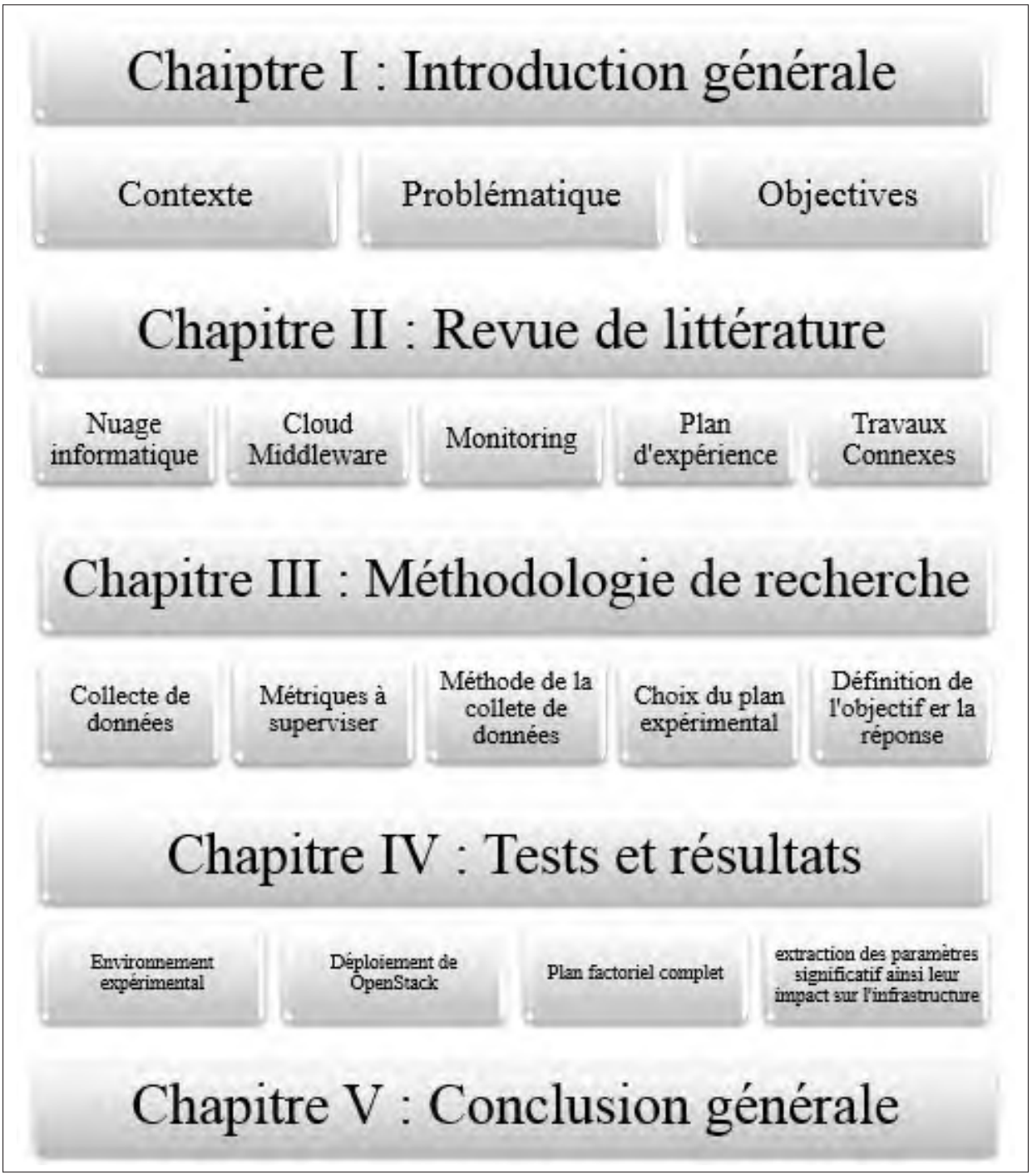

Figure 1.1 Diagramme des chapitres présentés dans cette mémoire

CHAPITRE 2

REVUE DE LITTÉRATURE

Ce chapitre présente une revue de littérature des différents aspects théoriques qui nous mènera à comprendre le problème de ce projet de recherche. En effet, ce chapitre s'appuie autour de quatre sections. D'abords, nous allons mettre le point sur le concept du nuage informatique, ses modèles de service, ainsi, nous présenterons le concept du Cloud middleawre puis, nous introduirons les différences plates-formes du Cloud Middleware existant dans la littérature et étant les plus pertinentes et les plus utilisées ; enfin, la dernière partie sera dédiée au profilage en général, nous focaliserons sur le profilage dans le Cloud et nous présenterons les travaux connexes de ce projet.

2.1 Nuage informatique

Le concept du Cloud computing s'appuie sur la délivrance des ressources informatiques. D'après le NIST1, il définit « l'informatique dans les nuages » en tant que modèle informatique offrant des ressources comme (réseaux, serveurs, stockages, applications, services) aux clients en fonction de leurs besoins, ce qui permet l'accessibilité : n'importe où, n'importe quand et par n'importe qui.

Ces derniers peuvent être partagés entre plusieurs clients à la fois et configurés selon leurs demandes. Cela nous ramène à un autre concept primordial dans le Cloud qui est la virtualisation. La virtualisation permet une gestion optimisée des ressources matérielles, autrement dit, avoir la possibilité d'exécuter plusieurs systèmes virtuels sur une même ressource physique partagée. Avec cette flexibilité, nous pouvons exploiter les ressources le maximum possible avec un effort minimal au niveau client.

2.1.1 Concept

Le nuage informatique est divisé en trois parties à savoir : le fournisseur (Provider), le client (Customer) et l'utilisateur (user).

Les fournisseurs offrent des services à l'utilisateur final en allouant les ressources physiques comme des centres de données de la technologie de virtualisation. Cela permet aux clients de commencer leur activité sans avoir à investir dans le matériel physique pouvant être acquis en le louant des fournisseurs.

Il existe trois modèles de services Cloud : Software as a Service (SaaS), Platform as a Service (PaaS) et Infrastructure as a Service (IaaS). Chaque modèle de services en nuage offre un niveau d'abstraction qui réduit les efforts requis par le consommateur de service pour créer et déployer des systèmes.

2.1.1.1 Infrastructure en tant que Service

The National Institute of Standards and Technology définit IaaS : La capacité fournie aux consommateurs est la capacité du traitement, le stockage, les réseaux et autres ressources informatiques fondamentales où le consommateur est en mesure de déployer et d'exécuter le logiciel donné, qui peut inclure des systèmes d'exploitation et d'applications. Le consommateur ne gère ou ne contrôle pas l'infrastructure Cloud sous-jacente mais il la contrôle via les systèmes d'exploitation, le stockage et les applications déployées.

Comme OpenStack est une infrastructure en tant que service, il assure le contrôle des grands bassins de calcul, de stockage et les ressources réseau à travers un centre de données, gérées par un tableau de bord ou via l'API OpenStack.

D'une manière technique IaaS donne un sens de son utilité dans des situations qui sont par-

ticulièrement adaptées dans l'infrastructure de Cloud :

– lorsque la demande est très volatile ; en tout temps il y a des pointes importantes et des creux en termes de demande sur l'infrastructure ;
– lorsque la demande est très volatile ; en tout temps, il y a des pointes importantes et des creux en termes de demande sur l'infrastructure ;
– pour les nouvelles organisations sans capital à investir dans l'infrastructure physique ;
– lorsque l'organisation est en croissance rapide et le matériel de mise à l'échelle serait problématique ;
– pour des raisons spécifiques, procès où les besoins en infrastructures sont temporaires.

Des exemples d'infrastructure en tant que Service, nous trouvons : Amazon web services, rackspace, IBM, Mirosoft, Vmware, etc.

Avec ce modèle, l'utilisateur s'intéressera plus à la conception et la gestion de ces applications, au lieu de s'inquiéter à gérer les centres de données et leur infrastructure. Fig.2.1 page 15.

2.1.1.2 Plate-forme en tant que Service

The National Institute of Standards and Technology définit PaaS : La capacité fournie au consommateur est de déployer les applications sur l'infrastructure Cloud consommatrice créée ou acquise à l'aide des langages de programmation, des bibliothèques, des services et des outils supportés par le fournisseur. Le consommateur ne gère pas ou ne contrôle pas l'infrastructure Cloud sous-jacente, y compris le réseau, serveurs, systèmes d'exploitation, ou le stockage, mais il les contrôle via les applications déployées.

D'un point de vue pratique, PaaS est particulièrement utile dans toute situation où plusieurs développeurs vont travailler sur un projet de développement ou lorsque d'autres parties externes ont besoin d'interagir avec le processus de développement. Aussi, PaaS est utile lorsque

les développeurs souhaitent automatiser les services de test et de déploiement.

La popularité de développement logiciel agile, un groupe de logiciels méthodologies de développement basées sur le développement itératif et incrémental, permettra également d'augmenter l'absorption de PaaS car il facilite les difficultés autour de développement rapide et l'itération de logiciels. Fig.2.1 page 15.

Des exemples de plate-forme en tant que Service : nous pouvons les subdiviser en quatre catégories, nous citons :

– Application frameworks : Google app Engine, CloudBees, IBM luemix, etc ;
– Big Data & storage : CLOUDANT, Amazon web services, platfora, etc ;
– Platform integrators : heroku, Joyent, Engine yard, etc ;
– Turnkey back-end : kinvey, Buddy, etc.

2.1.1.3 Logiciel en tant que Service

The National Institute of Standards and Technology définit PaaS : La capacité fournie aux consommateurs est d'utiliser les applications du fournisseur fonctionnant sur une infrastructure de Cloud. Les applications sont accessibles à partir de divers périphériques clients par le biais soit d'une interface client légère, comme un navigateur Web (par exemple, le courrier électronique basé sur le Web), ou une interface de programme. Le consommateur ne gère ni ne contrôle l'infrastructure Cloud sous-jacente, y compris le réseau, serveurs, systèmes d'exploitation, de stockage, ou même les capacités individuelles d'application, à l'exception possible des paramètres de configuration d'application spécifiques à l'utilisateur limité.

D'une manière technique, SaaS est une méthode de plus en plus rapide pour livrer la technologie. Cela dit, les organisations qui envisagent de passer vers le Cloud, vont envisager les

applications qu'elles déplaceront à SaaS. Comme tel, il existe des solutions particulières que nous considérons comme candidat pour un mouvement initial au SaaS ;

– applications où il y a des interactions significatives entre l'organisation et le monde extérieur. Par exemple, les logiciels d'email campagne de bulletin (newsletter campaign software) ;
– les applications qui ont un besoin important pour le Web ou l'accès mobile. Un exemple serait le logiciel de gestion des ventes mobiles ;
– le logiciel qui peut être utilisé pour un besoin à court terme. Un exemple serait le logiciel de collaboration pour un projet spécifique ;
– un logiciel des pics de demande de manière significative, par exemple l'impôt ou d'un logiciel de facturation utilisé une fois par mois.

Des exemples de logiciel en tant que Service, nous trouvons : salesforc.com, Amazon.com, microsoft, Google, etc.

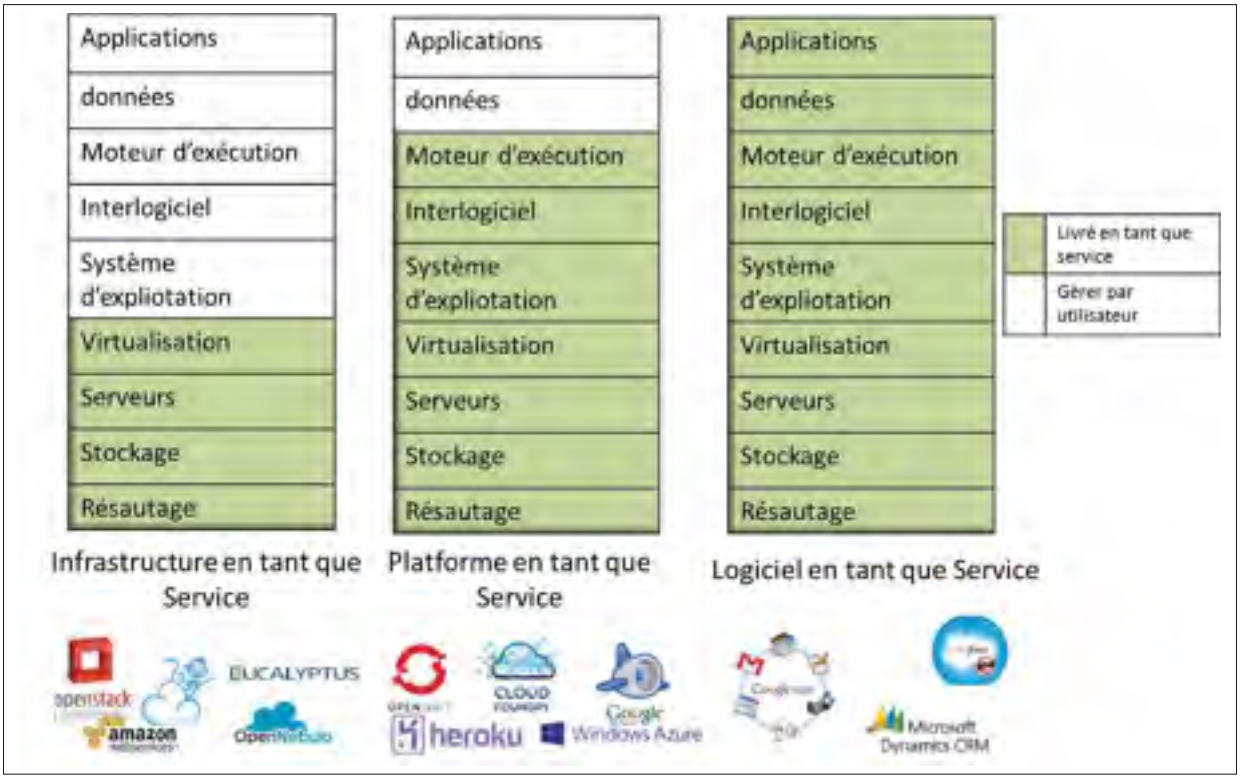

Figure 2.1 Différent modèle de service ainsi quelque application de chaque service

2.2 Cloud Middleware

Après avoir mis le point sur le Cloud computing, dans les sections précédentes, nous allons donner une brève définition du terme (Middleware) avant d'entamer le Cloud Middleware.

Le terme Middleware Cloud est un autre composant du nuage. Un logiciel qui est appelé interlogiciel (Middleware), est utilisé pour intégrer des applications de services et de contenus disponibles sur le nuage. Ce dernier est né avec l'apparition des systèmes distribués et se réfère à un ensemble de services standardisés tels que les APIs, les protocoles et les services d'infrastructure.

Ces derniers sont utilisés pour soutenir le développement rapide et pratique des services et des applications distribuées, basé sur le modèle Client/Serveur. Les principales caractéristiques clés du Middleware de nuage sont la gestion des données, l'identité, la sécurité, le service d'hébergement (hosting), de médiation et de gestion, le réseautage, l'interface utilisateur et de portails, la facturation le monitoring et la surveillance. La figure2.2 illustre que Middleware joue un rôle crucial sur l'intégration et l'interopérabilité des applications et des services fonctionnant sur des appareils informatiques hétérogènes. Il existe différentes solutions du Middleware Cloud, à savoir : OpenStack, OpenNebula, Nimbus, Eucalyptus et autre.

2.2.1 Eucalyptus

Eucalyptus Figure 2.3 était créé et conçu pour répondre au besoin de EC2 Cloud commercial. L'interface simple du service Web Amazon EC2 nous permet d'obtenir et de configurer des capacités avec un minimum d'efforts. Elle fournit un contrôle complet de nos ressources informatiques et nous permet d'exécuter notre application sur l'environnement informatique d'Amazon qui a fait ses preuves. Amazon EC2 réduit le temps requis pour obtenir et démarrer de nouvelles instances de serveurs à quelques minutes, ce qui vous permet de dimensionner rapidement la capacité, de l'augmenter et de la diminuer, au fur et à mesure des modifications de vos besoins de calcul. Amazon EC2 change l'aspect financier de l'informatique en permet-

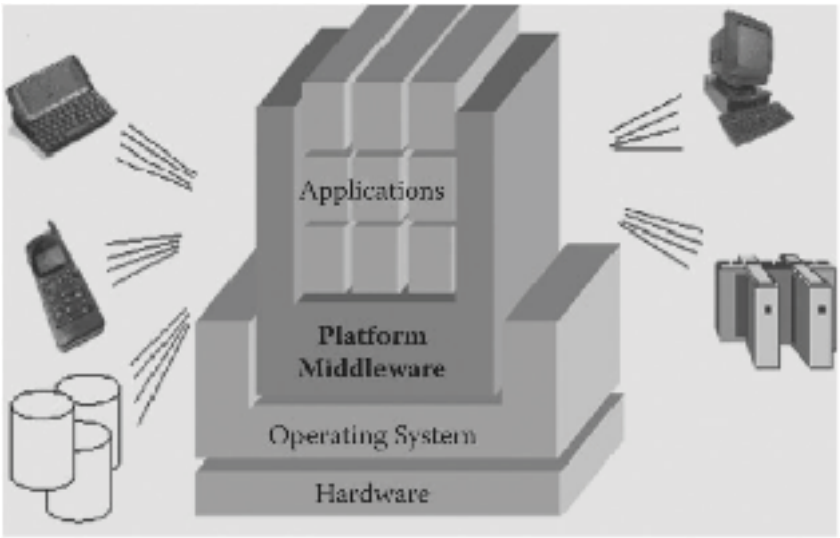

Figure 2.2 Abstraction de la couche du Cloud Middleware

tant de ne payer que pour la capacité que vous utilisez effectivement. Amazon EC2 fournit aux développeurs les outils pour construire des applications résilientes tout en évitant les scénarios de défaillance les plus courants (Amazon Web Services, 2015).

Eucalyptus s'appuie sur la décentralisation des ressources autant que possible, comme, le stockage utilisant un système appelé « Walrus » qui imite d'Amazon S3. Ce stockage est séparé en structure et distribué dans le Cloud. Ce mode de stockage permet aux virtuelles machines de s'exécuter indépendamment du nœud compute. Eucalyptus permet d'avoir des groupes multiples (un groupe de contrôleurs et un autre pour les interfaces des utilisateurs).

2.2.1.1 Provisionnement d'une instance

L'utilisateur utilise euca2ools pour demander une VM. L'image est chargée du Walrus (stockage virtuel) au nœud cluster Controller (CC) puis vers Node Controller (NC), ensuite un espace de block storage est associé à cette VM. Par la suite, le Node Controller (NC) joint un bridge pour joindre une carte réseau virtuelle (NIC) avec une adresse MAC virtuelle. Enfin, la VM est créée et prête à être utilisée, l'utilisateur peut y accéder via protocole SSH Figure 2.3.

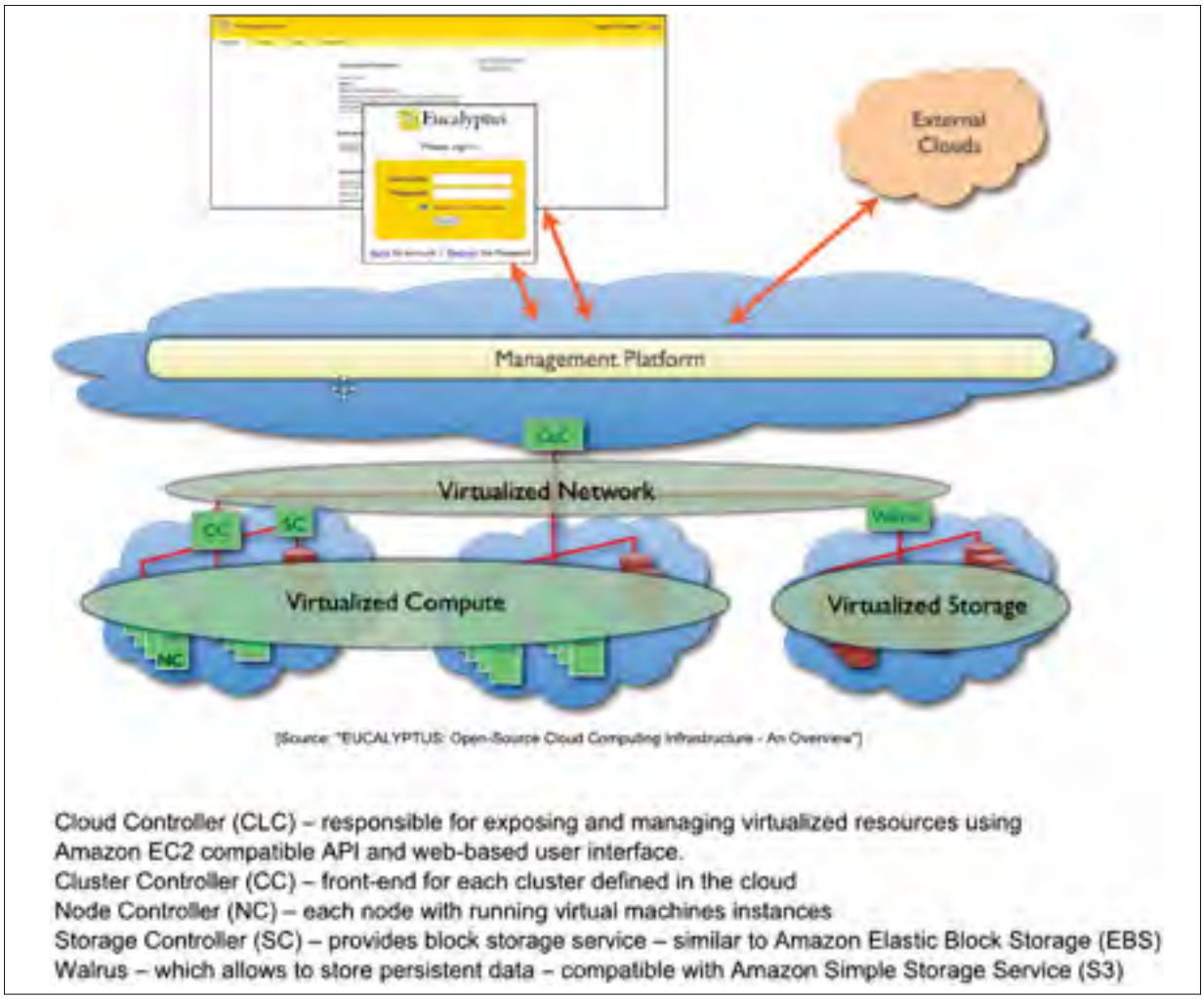

Figure 2.3 EUCALYPTUS : Open-Source Cloud Computing Infrastructure
An Overview

2.2.2 OpenNebula

OpenNebula Figure 2.4 se pose plus pour la centralisation et la personnalisation. Ils peuvent personnaliser le fichier de partage du système utilisé pour le stockage des fichiers du OpenNebula (par défaut ils utilisent NFS pour toutes les images et fichiers en exécution). Il permet la migration des VMs. De plus, le nœud compute requiert peu de ressources au niveau du disque dur vu que l'architecture d'OpenNenula est centralisée.

En revanche, cet accès à la personnalisation peut affecter la sécurité puisque le fichier du sys-

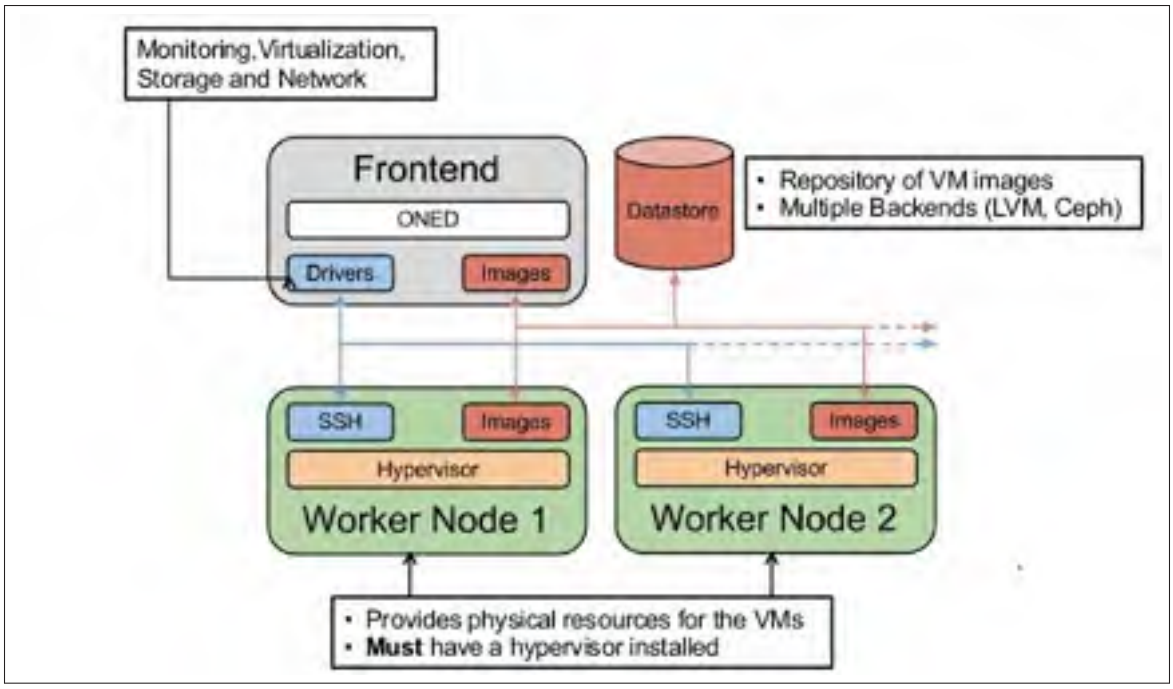

Figure 2.4 OPENNEBULA : Open-Source Cloud Computing Infrastructure - An Overview

tème NFS ne chiffre aucun trafic. Aussi, il exige de l'administrateur et l'utilisateur d'avoir des connaissances préalables pour tirer profit du système et pour ajouter d'autres caractéristiques par la suite, selon le besoin. Notons aussi que ce genre de Platform est conçu pour un nombre moyen d'utilisateurs connectés à ce type de Cloud. Ce genre de Cloud peut aider les chercheurs à exploiter d'autres axes de recherche en combinant ce type de Cloud avec d'autres technologies comme SGE ou Condor.

2.2.2.1 Provisionnement d'une instance

Premièrement, l'utilisateur utilise le protocole SSH pour se connecter au nœud de tête (cluster Node) pour faire une demande de VM. Deuxièmement, une copie du OS est associée avec une taille de disque à cette VM et configurée dans le répertoire NFS sur le nœud principal (Frontend Node). Troisièmement, le processus Node sur le nœud Frontend utilise ssh pour se connecter à un nœud compute dans le cluster Node. Quatrièmement, le nœud compute met en place un bridge de réseau pour fournir une carte réseau virtuelle (NIC) avec une adresse MAC virtuelle.

Cinquièmement, les fichiers nécessaires pour le hyperviseur sur le nœud compute seront tirés du NFS puis la VM est prête à l'utiliser. Enfin, l'utilisateur peut maintenant se connecter au VM via SSH directement.

2.2.3 OpenStack

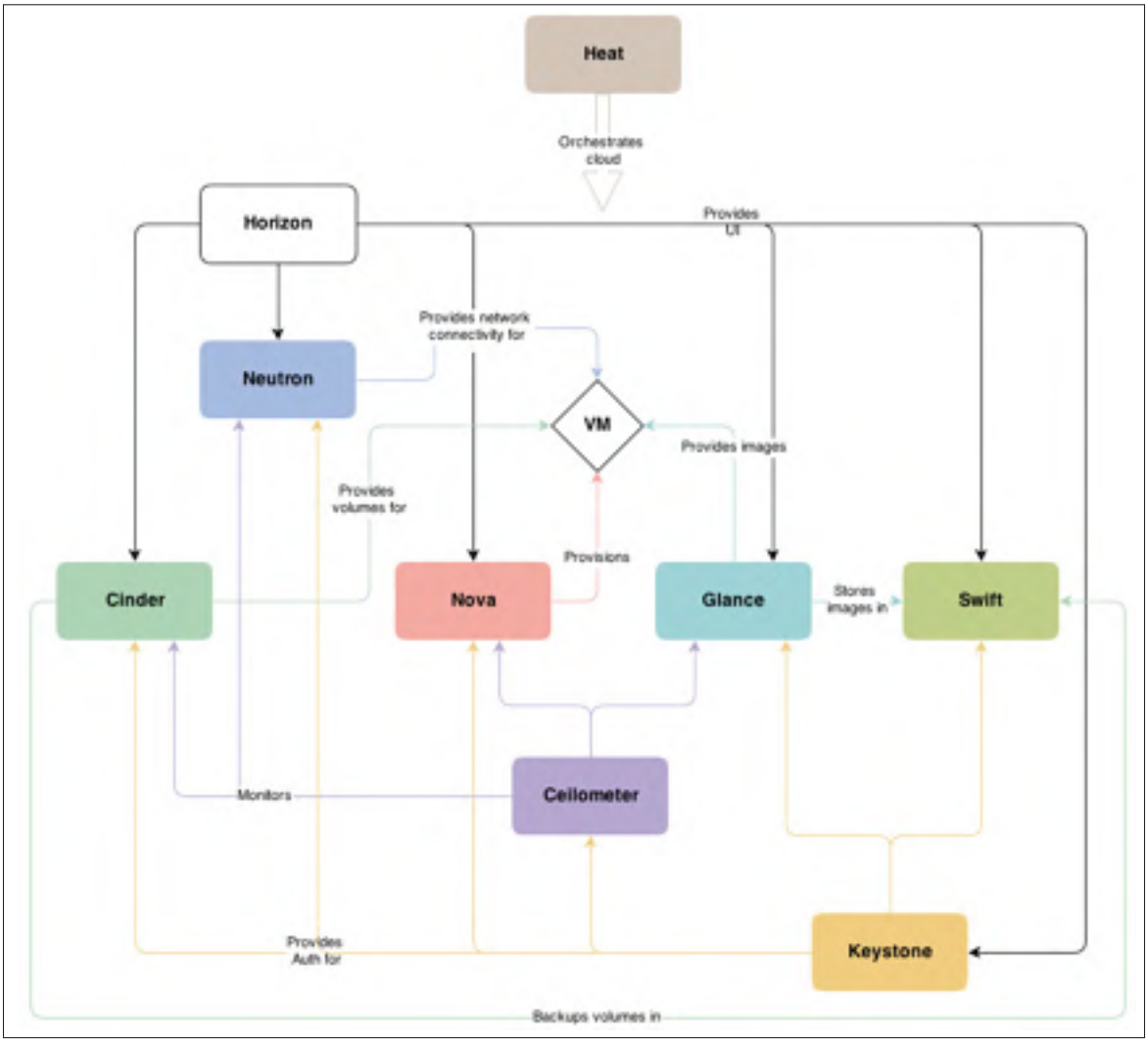

Figure 2.5 OPENSTACK : Open-Source Cloud Computing Infrastructure - An Overview

OpenStack est une collection de plusieurs composantes open-source qui assure les deux types de Cloud « privé et public ». Parmi ses composantes, nous trouvons : OpenStack Compute

(nommé Nova), OpenStack Object Storage (nommé Swift), OpenStack Image Service (nommé Glance). OpenStack prend de l'existence entre les compagnies grâce à son open source et au support de plusieurs compagnies.

Nova est conçu pour procurer et gérer le réseau virtuel des machines, créer une plate-forme de Cloud computing redondante et évolutive. Pour le Swift, il est utilisé pour créer un objet de stockage redondant et évolutif en utilisant des clusters de serveurs standardisés pour stocker des pétaoctets de données accessibles. Ce n'est pas un fichier de système ou système de stockage de données en temps réel, mais plutôt un système de stockage à long terme pour des données plus permanentes ou statiques. Glance garantit la découverte et l'enregistrement. Il associe les ressources nécessaires aux images des disques virtuels via le Service image (nous aurons plus de détails dans le quatrième chapitre).

2.2.3.1 Provisionnement d'une instance

L'utilisateur utilise Horizon (dashboard) pour s'identifier et demander une machine virtuelle. Keystone authentifie l'utilisateur et retourne l'entrée de l'instance avec la mise à jour du hôte ID approprié après filtrage lié à ce dernier. Nova prend les informations d'instance de la file d'attente et obtient les méta-données de l'image. Puis, il alloue et configure le réseau pour l'instance à une adresse MAC et IP. Ensuite, il obtient l'information de stockage par bloc. Enfin, NOVA génère les données pour le pilote de l'hyperviseur et exécute la demande sur l'hyperviseur (via libvirt ou api) Figure 2.5 (plus de détail dans le quatrième chapitre).

2.3 Monitoring

Le pilotage ou monitoring d'un système donné implique la mise en place d'un processus structuré afin d'évaluer leur performance. Il consiste à mesurer et à comparer les composantes du système lui même dans différentes circonstances.

En effet, quand nous parlons du monitoring, nous soulevons plusieurs questions, à titre d'exemple :

en quels sens nous nous focalisons pour mesurer la performance ? Comment et quels sont les facteurs adéquats ? Existe-il une base de référence qui aidera à qualifier la performance ? Quelles sont les explications des dérives et quels sont les indicateurs influents ? (Olivier Baude, 2010) ainsi, d'autres questions peuvent être soulevées et discutées !

En réalité, le monitoring touche divers champs de recherche, à savoir : le domaine des sciences et de la technologie, la médecine, l'écologie, les sciences sociétales, l'électronique et le domaine de l'informatique, ainsi que d'autres aspects de recherche. Dans notre recherche, nous nous intéressons plus au volet de l'informatique.

2.3.1 Monitoring dans le cloud

Du point de vu fournisseur ou/et consommateur, le Cloud monitoring est une opération très importante et primordiale. D'un côté, il est un outil clé pour le contrôle et la gestion des infrastructures matérielles et logicielles ; d'autre côté, il fournit d'une manière continue des informations et des indicateurs clés de performance (Key Performance Indicators (KPIs)) comme (disponibilité, délai, qualité service offert par le Cloud, consommation des ressources etc.) pour les plates-formes et les applications. Permettant à ces derniers de mettre en œuvre des mécanismes visant à prévenir (à la fois pour le fournisseur et les consommateurs).

En termes plus généraux, Cloud Computing implique de nombreuses activités pour lesquelles le monitoring est une tâche essentielle, voir figure 2.6. Dans la suite de cette sous-section, nous allons analyser attentivement les activités qui nous intéressent dans ce mémoire, en soulignant le rôle du monitoring.

À partir de la figure 2.6 nous avons plusieurs motifs qui nous poussent à faire le monitoring. Nous allons mettre le point sur trois besoins (facteurs) qui sont utilisés pour analyser la performance du Cloud Middleware, d'une part, et afin de tester Cloud Middleware qui est une couche essentielle dans le concept du Cloud Monitoring d'autre part.

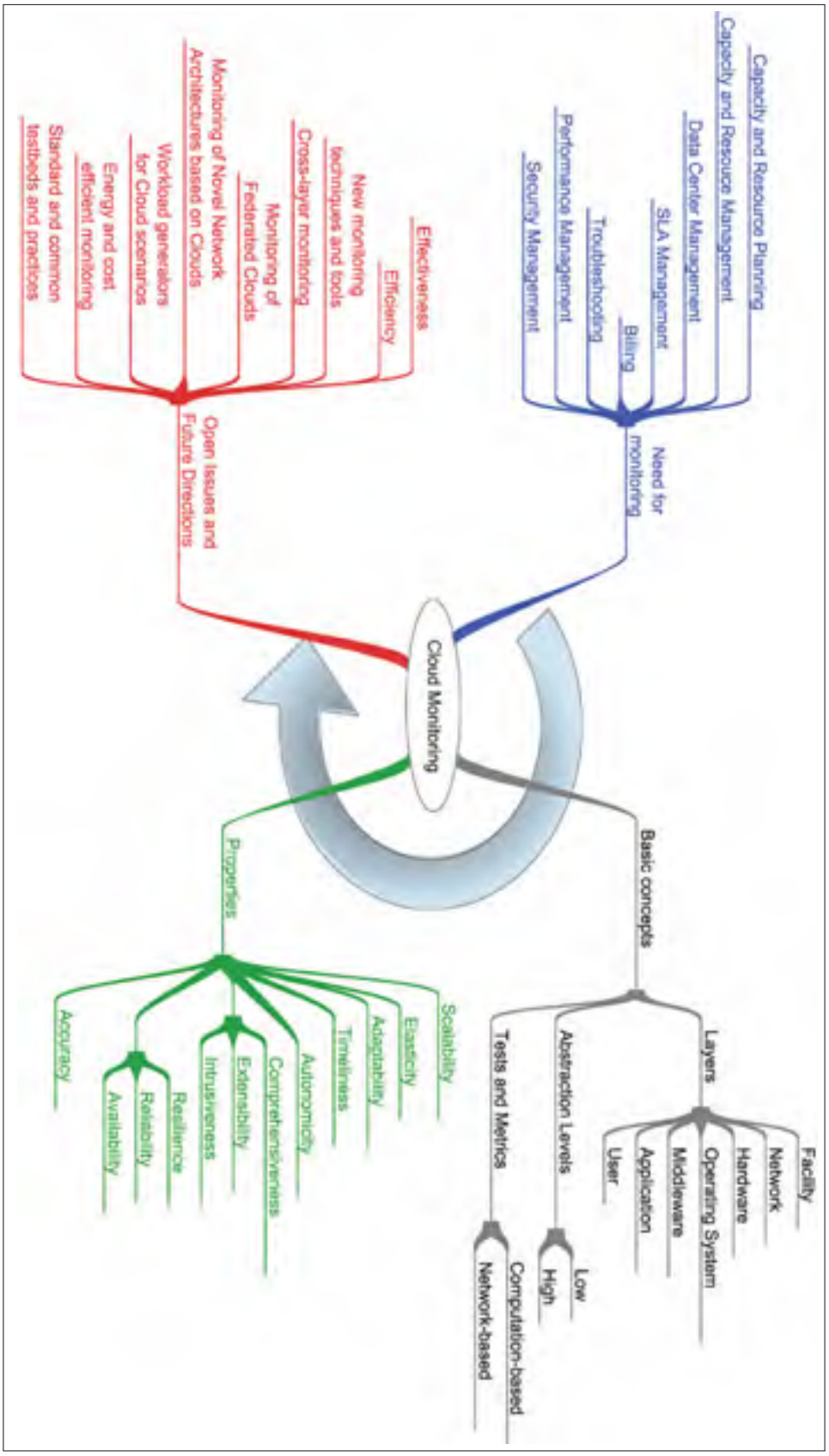

Figure 2.6 Cloud monitoring : les motivations, les propriétés, les concepts de base et les orientations futures
Tirée de Aceto et al. (2013)

2.4 Plan d'expérience

Dans le monde de la recherche, l'expérience est le seul moyen afin de valider les approches proposées ou d'arriver à une conclusion constructive. Nous pouvons définir une expérience comme un test ou une série d'essais dans lesquels les modifications délibérées sont apportées aux variables d'entrée d'un processus ou d'un système afin que nous puissions observer et identifier les raisons des changements qui peuvent être observés dans la réponse de sortie. Les plans d'expériences étaient introduits par le statisticien anglais Ronald A. Fisher dans les années 1930. À ses débuts, les premières modélisations étaient concentrées sur les domaines agricole et médical (chimie). Par ailleurs, à partir des années 80, les plans d'expériences furent une croissance exponentielle, touchant d'autres domaines comme les domaines de génies.

En effet, il existe deux classes des plans d'expérience : soit les Plans de surface de réponse (MONTGOMERY, 2001c), soit les plans factoriels (MONTGOMERY, 2001b), souvent leurs facteurs sont des variables continues et discrètes respectivement. Nous nous intéressons à la deuxième classe : les plans factoriels. Cette classe contient deux grandes catégories de plans : plans complets et plans factoriels fractionnaires.

Les plans complets 2^f ont un désavantage si nous considérons que le nombre f est supérieur à cinq, ce qui devient inapplicable et inefficient. La figure 2.7 explique que le nombre d'essais croit rapidement quand le nombre de facteurs augmente. De ce fait, dans certain cas, nous avons recours à des solutions alternatives comme les plans factoriels fractionnaires (MONTGOMERY, 2001d) afin de minimiser le coût (matériel, matière, personnel ..) et le temps de l'expérimentation. Souvent ces plans sont à deux niveaux 2^{f-p}. Par exemple, pour un plan à six facteurs nous nous rendons à 64 essais, dans le cas où nous négligeons les interactions triple et quadra, nous n'appliquons qu'une fraction des essais du plan complet. Si nous voulons la moitie du plan complet, le plan est dit 2^{6-1} ou le quart 2^{6-2} ou le huitième 2^{6-3} etc.

Bien choisir les essais signifie bien sélectionner les facteurs ce qui revient à l'utilisation du

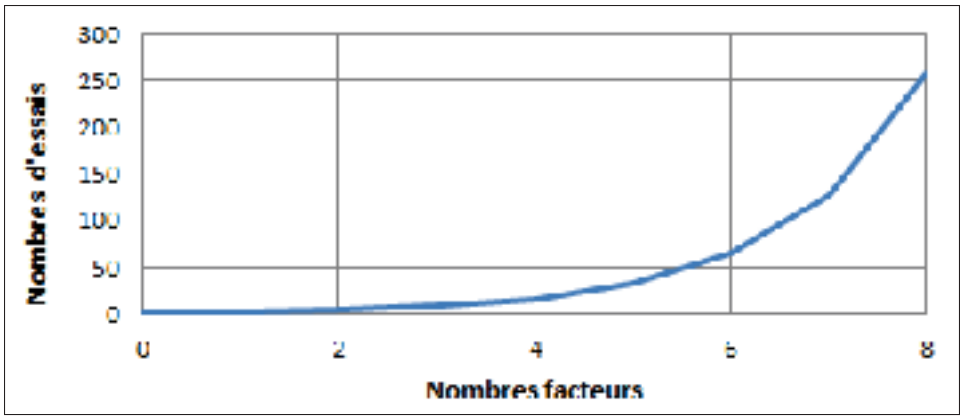

Figure 2.7 Nombre de facteurs en fonction du nombre d'essais

plan factoriel fractionnaire. Les plans factoriels fractionnaires sont des fractions des plans factoriels complets permettant de réduire le nombre d'essais à effectuer : d'où l'économie dans les expériences. Bien que nous réduisons l'espace des essaies, la qualité du résultat n'est pas garantie, mais si nous travaillons dans un cadre de contrôle et si nous choisissons bien les essais, cela peut être bénéfique.

2.5 Travaux connexes

Dans cette section, nous mettrons le point sur les travaux connexes que nous jugeons travaux potentiels pour notre mémoire de recherche et qui sont en relation avec notre problématique de recherche ; soit le profilage dans le soul qui englobe les métriques utilisées pour analyser la performance du Cloud, ainsi que son impact sur l'infrastructure du Cloud. Ces travaux tournent autour de l'analyse des performances, en termes de consommation des ressources dans les nuages.

2.5.1 Travaux connexes sur le monitoring dans le Cloud

Cette présente sous-section portera une analyse des articles qui ont soulevé la problématique de la performance des applications qui sont hébergées dans le Cloud.

En (Shao *et al.*, 2010), les auteurs ont proposé une méthode d'analyse, basée sur la corrélation canonique (CCA) afin de profiler les applications hébergées dans des machines virtuelles placées dans le Cloud, pour identifier la relation entre les performances des applications et les performances du système (consommation des ressources).

Puisque les besoins des ressources pour les machines virtuelles exécutant différentes applications sont dynamiques, alors, il est essentiel d'établir une relation entre la performance des applications en cours d'exécution à l'intérieur des machines virtuelles et la charge de travail dans un nœud physique ciblé. En d'autres termes, avant de placer la machine virtuelle sur le nœud physique, les auteurs ont pensé à modéliser une manière de prédiction de performances raisonnables qui aidera à prendre la décision en choisissant le nœud physique du Cloud qui va être cette machine virtuelle avec sa propre application.

L'idée des auteurs est d'associer un vecteur de poids canonique à chaque facteur qui représente le niveau d'implication (consommation des ressources) lié au système en corrélation avec les performances de l'application.

Les résultats montrent que le CCA est capable de trouver la relation entre le système et les performances des applications. Aussi, les tests prouvent que la vitesse de l'écriture I/O est le principal facteur qui est responsable de la performance des applications. Ils ajoutent qu'il y a une forte corrélation entre la performance des applications fonctionnant sur les machines virtuelles et les ressources matérielles des nœuds physiques. Ils concluent que les paramètres, avec un poids élevé dans le profil d'application, ont une précision de prédiction élevée pour être facteur clé pour la qualité de la performance.

Notre critique de cet article sera basée sur le fait que les auteurs se sont focalisés sur l'extraction de la relation entre l'application exécutée sur la machine virtuelle et la couche physique qui sont les ressources matérielles sans prendre en considération quel type de Cloud Middleware est utilisé. Autrement dit, ils ont négligé la présence du Cloud Middleware et que ce dernier

a aussi un impact sur la performance de la totalité du système : soit sur le niveau application, soit sur le niveau matériel. En effet, des travaux comme (Caron *et al.*, 2013) (Laszewski *et al.*, 2012), (Kurup *et al.*), (Ueda et Nakatani, 2010), (Sempolinski et Thain, 2010) et (Voras *et al.*, 2011) ont adressé le choix du Middleware comme une étape primordiale, car elle influence sur la performance des applications.De ce fait, la majorité de ces travaux ont favorisé l'utilisation du Cloud Middleware "OpenStack" sur les autres plates-formes (Eucalyptus, Nimbus et OpenNebula).

Dans (Zabolotnyi *et al.*, 2014) également, les auteurs ont mis en évidence le problème de sur-approvisionnement de l'utilisation des ressources physiques, par les applications, sans les utiliser d'une manière efficace, ce qui engendre des pertes : au niveau financier et/ou au niveau de gaspillage des ressources physiques. Afin d'optimiser l'utilisation des ressources, ils ont proposé une nouvelle approche de profilage fondée sur la planification des tâches uniformes *(*qui exigent un nombre de CPU et une capacité de mémoire RAM telles que les applications de traitement d'image satellitaire produisant 1 petabyte (PB) de taille d'image par seconde et qui sont prédéfinies par les développeurs avec l'utilisation des ressources non uniformes et les distribuent sur le nuage avec une connaissance préalable des limitations des ressources.

En effet, dans cet article, ils ont abordé la question de l'utilisation efficace et de l'élasticité des ressources du Cloud. Habituellement, les tâches exécutées dans le nuage n'utilisent pas les ressources (par exemple, mémoire, CPU ou de bande passante du réseau) d'une manière uniforme. Au lieu de cela, au cours d'exécution de la tâche, l'utilisation des ressources varie, provoquant des pics d'utilisation et des vallées. Afin d'obtenir des temps d'exécution efficaces et prévisibles, les développeurs ont à réserver des ressources en tenant compte de l'utilisation maximale que cette application peut atteindre.

Pour avoir des informations à jour et exactes sur le comportement de l'application dans chaque nœud physique (nuage hôte), l'application est surveillée par un système de profilage spécial. Le profiler est en cours d'exécution dans l'application sur chaque nœud de nuage utilisé et

recueille les informations nécessaires pour l'extension des décisions. Actuellement, cette information comprend l'utilisation de la mémoire par des sous-processus, en relation avec les tâches exécutées sur ce nœud et l'utilisation du processeur de la machine, afin d'éviter un éventuel chevauchement au niveau de l'agrégation de la charge en cas du pic.

L'approche consiste en deux phases : préapprentissage de la consommation des ressources en classant les tâches, puis les profils aux besoins des ressources. Après cela, une nouvelle tâche planifiée sera associée à la prédiction de l'utilisation des ressources qui est construite par la première étape. Ils comparent deux tâches avec et sans profilage basées sur la planification. Les résultats montrent que cette approche pourrait économiser 33 % de mémoire et 1 % de temps pour terminer la même tâche.

Notre critique de cet article est la suivante : ils se sont concentrés sur un sous-ensemble spécifique d'applications Cloud Computing qui consistent à utiliser un ensemble de taches uniformes dans un ensemble de ressources physiques non uniformes. De ce fait, nous pouvons dire que cette approche ne pourra pas être généralisée pour une application quelconque .Aussi, cette approche vise la troisième couche Cloud Computing qui est la couche application service, ceci ne tient pas en compte qu'il y a aussi une influence de couche infrastructure qui a pour rôle d'orchestrer, d'acheminer les taches au nœud approprié et surtout de contrôler tous le système du Cloud.

Dans cet article, (Mian *et al.*, 2013), ils examinent le problème de l'approvisionnement des ressources dans un Cloud public afin de traiter l'analyse des données reliées à des charges de travail. Ils ont développé une méthode de provisionnement qui a pour but de déterminer quelle est la configuration la plus rentable pour une charge de travail d'analyse de données. Ils considèrent qu'une configuration rentable est celle de telle sorte que les coûts des ressources soient réduits au minimum, tandis que les Service Level Agreements (SLA) associés à la charge de travail soient au maximum.

Ils ont proposé une formulation du problème de provisionnement basé sur une méthode heuristique avec deux variantes (heuristique gloutonne et heuristique gloutonne adaptée) en vue de définir un cadre pour résoudre le problème. Ce cadre de travail contient un modèle de coût pour prédire le coût d'exécution d'une charge de travail sur une configuration et un procédé de sélection de configurations la plus adéquate. Le modèle de coût tient en considération à la fois, les coûts des ressources et les pénalités de SLA. Les demandes et les fréquences de ressources spécifiées sont comptabilisées en fil d'attente des modèles de réseaux de machines virtuelles, qui sont utilisés pour prédire la performance.

Cette approche, en matière de provisionnement des charges de travail d'analyse de données d'échantillons, était testée sur la plate-forme Amazon EC2. Les prévisions de coûts de ce modèle pour les charges de travail de l'échantillon sont à 7 % des coûts réels sur EC2. Ils montrent également que cette approche offre plusieurs configurations rentables pour les charges de travail de l'échantillon que la méthode manuelle provisionnement utilisée par (Vázquez-Poletti *et al.*, 2012) est basée sur l'observation qui négligent plusieurs facteurs comme les frais de stockage ou de communication appliqués à une charge de travail.

Nous finissons cette première partie de la revue de littérature avec ce dernier article qui traite le sujet d'analyse de la performance du Cloud pour l'utilisation des applications scientifiques qui est gourmande en terme de calcul.

Le Cloud promet d'être une alternative des autres modèles comme grappes, grilles, et les superordinateurs. Cependant, la virtualisation peut induire des pénalités de performance significatives pour le calcul scientifique, exigeant les charges de travail. Dans ce travail, ils ont présenté une évaluation de l'utilité des services de Cloud Computing actuelle pour le calcul scientifique. À vrai dire, ils ont choisi EC2 comme banc d'essai, leur travail s'est focalisé sur la question suivante : la performance de nuages, est elle suffisante pour le calcul scientifique ?

À cette fin, leur méthodologie de recherche s'est accentuée sur deux parties, la première est

le Cloud spécifique, la deuxième est l'infrastructure agnostique. Une caractéristique du nuage est qu'il y a toujours des ressources inutilisées, afin qu'ils puissent être obtenus à tout moment sans temps d'attente supplémentaire. Cependant, la charge d'autres systèmes à grande échelle (grilles) varie au fil du temps, en raison des habitudes de soumission ; nous voulons étudier si de gros nuages peuvent, en effet, contourner le problème de la charge à grande échelle (grilles) qui varie au fil du temps. Ainsi, nous testons la durée d'acquisition des ressources et la libération sur des périodes courtes et longues avec différents types d'instances, afin d'examiner la performance du CPU I/O et la mémoire.

Ils constatent que la performance est reliée au I/O. Les instances de type x.large ont de meilleures performances d'I/O de tous types d'instances. Ceci revient à la taille du cache qui joue un rôle très important, ainsi, plus la taille du cache est grande et plus la performance est meilleure. La principale conclusion est que la performance et la fiabilité du nuage testé sont faibles. Et le nuage est insuffisant pour le calcul scientifique dans son ensemble, même si elle fait toujours appel aux scientifiques qui ont besoin de ressources immédiatement et temporairement.

D'une part, la majorité des travaux ont porté sur de hauts niveaux, la couche d'application qui est construite sur le haut du modèle d'architecture, en analysant le suivi des performances des applications, en utilisant plusieurs paramètres comme I/O, CPU, RAM, la consommation d'énergie, l'instrumentation code, en extrayant le comportement anormal, à partir de l'historique du log et autres.

2.5.2 Travaux connexes sur le Cloud Middleware

D'autre part, au cours de ces dernières années, plus précisément ces quatre dernières années, plusieurs travaux ont fait le sujet de traiter, d'évaluer et d'analyser le coud Middleware OpenStack qui fait notre Middleware choisi par la suite. Vu ses points forts et sa grande utilisation dans la communauté des Cloud libres open-source, nous allons présenté six diffèrents travaux qui ont été menés pour étudier OpenStack sur des aspects variés.

En (Corradi *et al.*, 2014), leur but était de présenter une infrastructure de gestion de Cloud ainsi que de fournir des conseils utiles à la communauté OpenStack pour la conception et la mise en œuvre des mécanismes de gestion Nuage de nouveaux.

L'architecture du Cloud exploite des techniques de virtualisation afin de provisionner plusieurs machines virtuelles (VM) sur le même hôte physique, pour utiliser les ressources disponibles d'une manière efficace. Par exemple, consolider les machines virtuelles dans un nombre minimal de serveurs physiques pour réduire la consommation d'énergie de l'exécution. La consolidation d'une VM doit examiner d'une manière précise la consommation des ressources en agrégats de co localisé VM, afin d'éviter une baisse des performances et la violation du Service Level Agreement (SLA).

En effet, VM consolidation soulève également plusieurs questions de gestion, car il tend à une exploitation optimale des ressources disponibles, tout en évitant une grave dégradation des performances, en raison de la consommation des ressources co localisée VM. Les auteurs ont proposé une solution de gestion de consolidation des VM pour OpenStack. En outre, ils proposent une plate-forme de gestion Cloud pour optimiser la consolidation des VMs en tenant en compte, trois dimensions principales, à savoir la consommation d'énergie (visant à traiter le volet environnemental et économique), les ressources de l'hôte (optimiser les performances d'exécution), et le réseautage (optimiser l'utilisation du réseau de centre de données "bande passante et de latence"). La raison pour laquelle, ils ont abordé les trois aspects à la fois vu que dans leur revue de littérature, chaque problème a été traité indépendamment et en négligeant les autres aspects.

Leur design de leur infrastructure de gestion cloud se compose de trois éléments (étapes) :

– première étape : recueillir des données du nuage ;
– seconde étape : prendre soin d'utiliser les données recueillies pour calculer un nouveau et plus approprié placement VM, si disponible ;

– troisième étape : calculer le plan de déplacement de VM et appliquer les changements.

Ainsi, l'infrastructure de gestion proposée est constituée d'un composant de surveillance, un placement calcul de gestionnaire, et un actionneur de placement, disposé dans une canalisation où chaque étape exploite la sortie de la précédente. La composante de surveillance rassemble à la fois le système et le niveau de service d'informations sur les ressources utilisées, et les rend disponibles à l'étape suivante.

Le placement calcul gestionnaire exploite l'information de l'utilisateur SLA surveillance pour vérifier si un meilleur placement VM existe ; il contient trois sous-modules pour traiter explicitement avec l'énergie, les ressources de l'hôte, et le réseau. Enfin, si un meilleur placement existe, l'actionneur Placement prend soin d'exécuter les opérations VM plan de migration et de reconfiguration.

Leur expérimentation tourne autour de trois exemplaires et études de cas :

– la première, calculer la consommation de VCPU, vise à souligner la dégradation des performances lorsque plusieurs machines virtuelles co-localisées exécutent des tâches de calcul lourd ;
– la seconde, le fournisseur de serveur HTTP Apache, estime que les VMS hébergent les serveurs HTTP Apache ensuite analysent les effets secondaires possibles lorsque de nombreux serveurs Web co-localisés traitent plusieurs demandes entrantes ;
– la troisième, la communication du réseau vise à souligner les problèmes de performance introduits par la virtualisation lorsque VM effectue des transferts de données très élevés, exemple les services de MapReduce.

Ils concluent que la consolidation VM introduit de nouveaux défis qui doivent être considérés par l'infrastructure de gestion Cloud. Bien que la colocalisation des machines virtuelles sur le même serveur physique est toujours recommandée en termes d'économies d'énergie, il

peut conduire à des effets secondaires de la performance ; ceci dépend strictement du type de service géré. D'une part, les services de calcul sont plus faciles à gérer, car la dégradation de rendement résultante peut être facilement prédite. Dans le cas de surcharge du CPU, toutes les machines virtuelles recevront une bonne quantité de CPU.

D'autre part, les services de réseau liés sont plus difficiles à gérer, car la dégradation de la performance est une conséquence d'un mélange de la charge de CPU et du réseau, où le second affecte également la charge de la CPU du serveur physique. Pire encore, la dégradation des performances en raison de la mise en réseau dépend, vraiment, de la plate-forme de virtualisation et des pilotes de périphériques adoptées.

En (Xu et Yuan, 2014) évalue la performance du OpenStack plus précisément le composant Quantum. En effet, le réseau virtuel est une partie importante de l'informatique en nuage. Quantum est l'une de la plate-forme de gestion du réseau virtuelle du nuage le plus populaire. Actuellement, la composante Quantum a deux plans de déploiement qui est le plan à un hôte unique et le plan à multi hôtes. Les auteurs ont présenté une analyse du mécanisme du fonctionnement de la composante Quantum.

Dans l'article, ils ont introduit le problème de la dissolution du nœud réseau dans le cas du plan à hôte unique. La communication interne et externe de toutes les machines virtuelles dans les nœuds de calcul est réalisée par les nœuds du réseau, de sorte qu'il provoque probablement la congestion de nœuds de réseau. Ce qui peut causer le disfonctionnement du réseau, impliquant directement l'effondrement du Cloud. Afin de résoudre les problèmes des scénarios de déploiement à unique hôte, ils ont conçu un ensemble de régimes multi hôtes de OpenStack, qui se compose principalement d'un nœud de contrôle et 10 nœuds de calcul, ces nœuds forment trois réseaux, qui sont gestionnaires du réseau, réseaux de données et réseaux externes.

Pour les expérimentations, ils ont créé un réseau virtuel et dix machines virtuelles avec le système d'exploitation du serveur d'Ubuntu dans le OpenStack et veillent à ce qu'ils sont en

mesure de communiquer, puis déployer des clusters Hadoop sur ces serveurs virtuels. Clusters Hadoop se compose d'un nœud maître (gestion des tâches) et plusieurs nœuds esclaves (traitement des données) avec cinq types de données de tailles différentes. Comme métrique de performance, ils ont pris "map time" et "reduce time of Hadoop". Ils ont mis en place trois tests ; le but du premier est d'analyser les performances du réseau sur le nœud physique. L'expérience a prouvé qu'un seul nœud physique a une capacité suffisante pour gérer ces communications. Le deuxième test est de traiter les performances du réseau en mode multi hôtes. Les résultats des expériences montrent que les performances du réseau virtuel du plan multi hôtes ont été grandement améliorées pour comparer le plan hôte unique.

Par ailleurs, les auteurs de l'article (Callegati *et al.*, 2014b) ont étudié la performance du multi locataire des réseaux virtuels dans les infrastructures de Cloud (OpenStack). En effet, les architectures de réseaux futurs basées sur le Network Fonction Virtualization émergents et le Software Defined Network paradigmes reposeront essentiellement sur les infrastructures de Cloud Computing, en profitant des technologies de virtualisation. La performance des réseaux virtuels multi locataire sera alors un élément clé à prendre en considération, lors de la conception de telles architectures basées sur le Cloud Ce dernier apporte de grands avantages en termes de gestion de réseau flexible effectuée au niveau du logiciel et la coexistence possible de plusieurs clients partageant la même infrastructure physique.

Afin d'évaluer la performance du réseau virtuel assurée par OpenStack, il faut que plusieurs locataires soient actifs simultanément. Ils ont déployé un petit banc d'essai comprenant un nœud de contrôleur, un nœud de calcul et un nœud de réseau. Le nœud de calcul fonctionne KVM, et est équipé de 8 Go de RAM et un processeur quad-core permis multi-thread, résultant en 8 processeurs virtuels. Une configuration de Cloud OpenStack a été considérée, l'hébergement de plusieurs locataires d'exécution d'une architecture de réseau virtuel NFV-inspired simple dans le nuage. L'étude de cas considérée est une chaine d'inspection approfondie des paquets "deep packet inspection (DPI)".

Comme métrique de performance, ils ont choisi le taux de paquets et le débit reçus de sources de trafic en cours d'exécution dans chaque client VM. Ils ont généré des paquets à taux croissant (RUDE,CRUDE tool), pour les différents nombres de locataires activés simultanément. Les résultats montrent que le réseau virtuel n'est pas en mesure d'utiliser toute la capacité physique, même dans le cas d'un seul locataire. Cette pénalité augmente avec le nombre de locataires (avec la complexité du système virtuel). Aussi, l'augmentation d'une unité du nombre de locataires entraîne une diminution d'environ 10 % de la capacité de réseau utilisable.

Une autre étude similaire aux (Callegati *et al.*, 2014a) a été menée par les mêmes auteurs qui se sont intéressés à la performance d'OpenStack dans des conditions critiques en évaluant le taux maximum de paquets. Ensuite, ils ont isolé la performance des principaux composants du réseau d'OpenStack et déterminé où se trouvent les goulots de saturation, en spéculant sur les améliorations possibles. Pour cette fin, ils ont construit deux bancs d'essai, à savoir OpenStack scénario (utilisant Linux Bridge) et le scénario non-OpenStack (utilisant OVS), avec KVM comme natif de Linux VM Hypervisor. Pour chaque banc d'essai, plusieurs montages expérimentaux ont été configurés (réalisation avec une seule compute Node et avec deux compute Node pour les deux scénarios), où une source de trafic envoie des paquets à un taux croissant vers une destination qui mesure le taux de paquet reçu et le débit. L'outil RUDE et CRUDE ont été utilisés pour la génération et la mesure du trafic.

À partir des résultats, ils ont conclu que la mise en œuvre du réseau virtuel standard d'OpenStack peut montrer les limites de performance significative. Et que OVS affiche toujours une meilleure performance que Linux Bridge. La performance se dégrade quand nous augmentons la taille des paquets en utilisant Linux Bridge. Cela prouve que la présence de ponts supplémentaires (Linux Bridges) dans les nœuds de calcul est une des principales raisons de la dégradation des performances d'OpenStack.

En (Scharf *et al.*, 2015) ils présentent une extension d'ordonnancer OpenStack qui permet un placement de réseau courant en prenant en compte les contraintes de bande passante vers et

à partir des nœuds. Leur solution assure le suivi de l'allocation des ressources de réseau hôte local, et il peut être combiné avec des mécanismes d'application de bande passante telle que la limitation de débit. Ils présentent un prototype qui nécessite du changement dans le logiciel open source OpenStack. L'ordonnancer par défaut ne considère que les ressources de calcul et de mémoire, mais il ne tient pas compte des exigences de bande passante du réseau.

La solution la plus simple sur le plan conceptuel est d'améliorer la planification d'affinité en considérant la localité. Grâce à la connaissance nœud/rack, le planificateur tente de placer un ensemble d'instances proches les unes des autres, sur le même nœud ou à l'intérieur du même rack. Mais, cette approche ne donne aucune garantie sur la performance du réseau, et il ne propose aucun moyen de distinguer les différentes exigences en matière de mise en réseau des instances.

La bande passante de réseau d'un nœud est allouée par l'ordonnancer de la même façon que les ressources de calcul et de mémoire. Lors du démarrage d'une nouvelle instance, une certaine bande passante doit être demandée de la même manière comme CPU virtuel, RAM, etc. Dans ce cas, une instance peut demander une bande passante totale sur l'hôte sans spécifier une matrice de trafic plus détaillée. Toutefois, ce modèle ne prend pas en charge les demandes qui garantissent la bande passante de bout en bout, si nous adressons le problème de qualité de service, mais cela n'est pas leur but dans cette recherche.

L'évaluation des algorithmes et des politiques d'une manière exhaustive directement sur un nuage est impossible, car il implique beaucoup de temps et d'efforts. En (Sitaram *et al.*, 2014) ils ont présenté un simulateur OpenSim des services OpenStack. Il a été développé avec l'intention de simuler de nouveaux algorithmes et aussi de permettre une étude de l'utilisateur, du comportement d'une application fonctionnant sous différentes configurations de déploiement dans un environnement OpenStack. OpenSim est développé au-dessus de CloudSim, et un simulateur libre est basé sur les événements.

Un des principaux objectifs du simulateur OpenSim est d'aider l'utilisateur à comprendre comment un nuage de OpenStack déployé sans entreprendre les processus de déployer dans un monde réel. En outre, il permet également à un utilisateur d'exécuter à plusieurs reprises et de mener une série d'expériences de simulation avec une légère variation des paramètres d'une manière rapide et facile. OpenSim génère également des informations sur le temps de réponse des requêtes, temps nécessaire pour traiter les demandes et d'autres mesures. Les résultats de simulation montrent qu'OpenSim peut modéliser correctement un environnement OpenStack et peut prédire avec précision le débit et le temps de réponse sous différentes charges de travail.

Ce mémoire traite l'impact de la charge de travail sur la performance du Cloud Middleware ainsi que l'impact sur la consommation des ressources physiques. À la différence des autres travaux, nous nous intéressons plus à la performance de Cloud middleawre tel que la performance des applications s'exécute sur ce Cloud Middleware ou s'analyse d'une seule composante du Cloud Middleware.

2.6 Conlusion

Dans ce chapitre, nous avons mis le point sur le concept du nuage informatique, qui se positionne actuellement parmi les nouvelles technologies basées sur la virtualisation, le partage des ressources informatiques. Le nuage informatique garantit l'extensibilité et l'élasticité et l'accès à ces ressources à la demande. Aussi, le nuage informatique assure l'exploitation des ressources par un grand nombre d'utilisateurs et sont facturées en fonction de l'utilisation.

De plus, nous avons présenté le Cloud Middleware comme un logiciel qui assure l'intégrité des applications des services et du contenu disponibles sur le nuage informatique, ainsi que son utilité dans la gestion des données, la sécurité, le service d'hébergement, le monitoring et d'autres taches. Nous avons illustré trois exemples de Cloud Middleware.

Nous avons clôturé ce chapitre par une revue littéraire des travaux connexes sur les deux as-

pects d'utilisation des applications dans le Cloud ainsi que leur impact sur la consommation des ressources et une revue sur le Cloud Middleware, plus précisément *OpenStack* pour savoir et analyser les travaux qui ont été faits sur ce dernier.

CHAPITRE 3

MÉTHODOLOGIE DE TRAVAIL

Au cours de ce chapitre, nous allons répondre aux problématiques citées dans la section 1.3. Après avoir présenté un aperçu sur monitoring dans le Middleware Cloud dans la section 2.2, le premier bloc de ce chapitre présentera les métriques appropriées décrivant la performance du Cloud Middleware. Ainsi, un outil sera utilisé pour collecter les données à moindre coût. Puis, nous allons modéliser et optimiser la performance du Cloud Middleware d'une manière efficace. Cette solution nous aidera à analyser et calculer l'impact des ressources physiques sur la performance du Cloud Middleware.

3.1 Collecte de données

D'une manière générale, nous pouvons subdiviser le Cloud Middleware en trois grandes parties qui définissent ce dernier ; à savoir, le nœud du contrôleur (Controller Node), le nœud de calcul (Comptue Node) et le nœud du réseau (Network Node). Notre but de cette présente recherche est de profiler le Cloud Middleware, mais nous nous intéressons plus précisément au nœud Contrôleur. Dans cette présente section, nous allons définir toutes les métriques qui sont reliées à la performance du Cloud Middleware, dans le but de répondre à la *première question* de la problématique soulevée dans la section 1.3.

3.1.1 Les métriques à superviser

Afin de profiler le nœud contrôleur du Cloud Middleware, l'étape du choix des paramètres est cruciale pour baser notre étude. Étant donné un Cloud Middleware, nous avons plusieurs paramètres qui sont soit contrôlables soit incontrôlables par l'administrateur.

Quant à notre intérêt, nous avons plus tendance vers le premier critère (contrôlable). Nous rappelons ici, que l'importance des objectifs du Cloud Middleware est de nous offrir un ensemble de machines virtuelles appelé instances aux utilisateurs et accessibles de n'importe quel lieu.

De cet effet, à chaque instance créée, nous aurons besoin de spécifier les paramètres suivants :

– utilisateur (user) : Chaque instance créée est utilisée par un utilisateur ;
– locataire (tenant) : Dans chaque locataire, nous trouvons une ou plusieurs instances qui sont accédées par un ou plusieurs utilisateurs ;
– image : Chaque instance contient un Système d'exploitation ;
– stockage : Chaque instance a un disque dur attaché ;
– CPU et RAM : Chaque instance a un nombre de CPU et une capacité de RAM ;
– réseau : Chaque instance a une adresse IP et une adresse Mac ; pour être accessible de l'extérieur ;
– instance(s) : Nombre d'instances à créer.

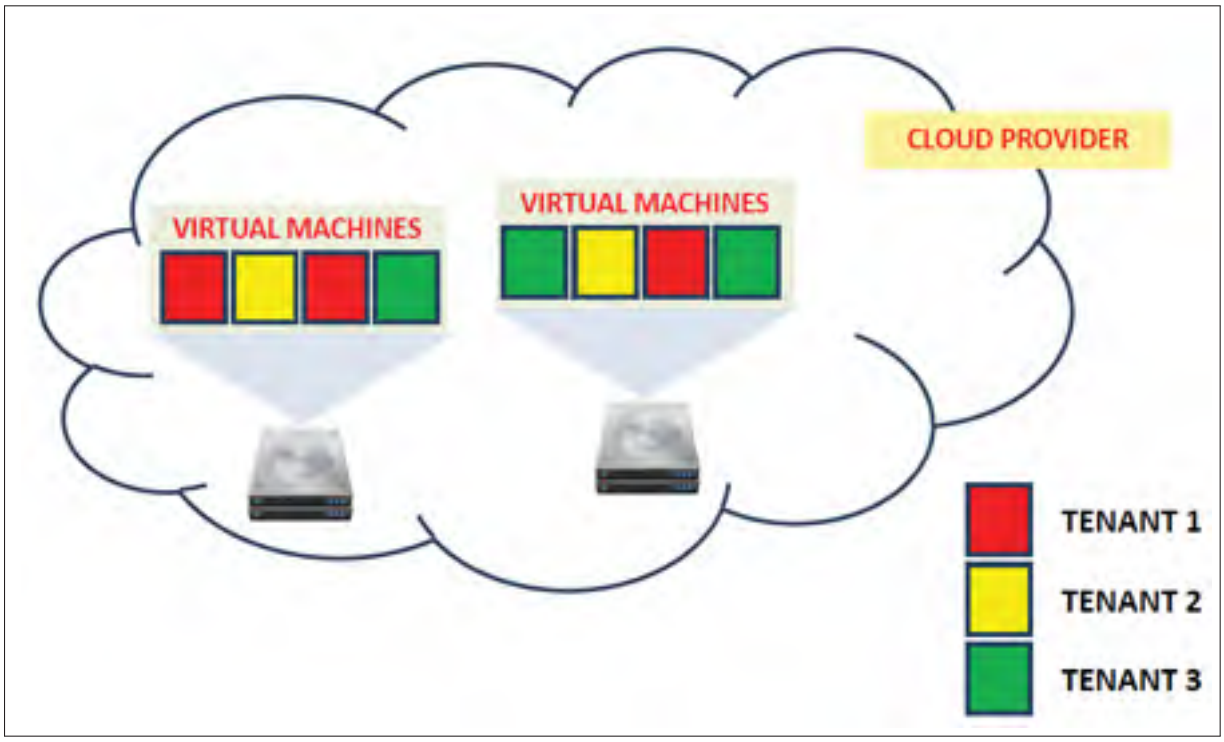

Figure 3.1 Exemple de deux nœuds compute contenant des machines virtuelles appartenant à des multi locataires

Dans le but de normaliser les expériences, nous avons opté de fixer les paramètres (type image, CPU et RAM) de chaque instance créée au prix d'avoir des résultats objectifs. En échange, nous avons choisi la création et la suppression des instances, comme scénario principal, vu son importance dans le modèle du Cloud Middleware. Car sans les instances, nous ne pourrons rien faire dans un Cloud Middleware. Aussi, il va falloir déterminer s'il existe une relation entre ces trois facteurs (locataire, utilisateur et instance).

Arrivés à ce stade-ci, nous supposons que nous avons bien défini nos facteurs significatifs (plus d'explication dans la section 3.2). Maintenant, ce qui nous reste pour compléter la liste appropriée des paramètres de performance, c'est d'identifier les métriques à collecter durant la phase des tests afin de construire une synthèse sur la variation de la consommation des ressources pendant les différentes charges de travail appliquées. Mais, avant de lister les métriques que nous avons choisies et expliquer pourquoi nous nous sommes limités juste à ces métriques. Nous devons éclaircir le point suivant.

En effet, le premier critère, que nous allons utiliser pour tirer une conclusion solide, est le temps et plus particulièrement, le temps d'exécution nécessaire pour accomplir le scénario choisi, ainsi que le temps de réponse (latence). Avec ces deux paramètres, nous pouvons répondre aux questions suivantes :

– pour une charge de travail donnée, combien de temps prendra-t-il pour exécuter cette requête ?
– combien de temps faut-il pour traiter la requête ?
– quel est le taux moyen d'exécution des différentes charges de travail ?

Le choix de ce premier critère a été soulevé comme métrique pour analyser la performance du Cloud Middleware et a été utilisé dans des travaux de recherche antérieurs. Nous trouvons le temps d'exécution parmi toutes les métriques de la performance discutées dans les travaux connexes dans la section 2.3 étant une métrique très importante.

Mis à part le temps nous avons CPU étant une autre métrique qu'il faut prendre en considération. En général, la représentation de CPU est sous un pourcentage moyen d'utilisation du CPU, ce pourcentage indique le temps d'inactivité du CPU qui a été soustrait de 100 pour cent. Par ailleurs, il existe une variété de manières, comment pouvons nous calculer le taux de CPU utilisés, nous citons ;

– CPU Time : le temps passé à exécuter le programme cible sans tenir compte du I/O ou l'exécuter d'autres programmes ;
– User CPU time : Le temps passé à exécuter les lignes de code qui sont "dans" notre programme ;
– Cycles d'horloge : c'est une opération de matériel numérique régie par un débit constant de l'horloge (voir figure) aussi, au lieu de rapporter temps d'exécution en secondes, on utilise souvent les cycles $\frac{seconds}{program} = \frac{clock\ cycles}{program} X \frac{seconds}{clock\ cycle}$.

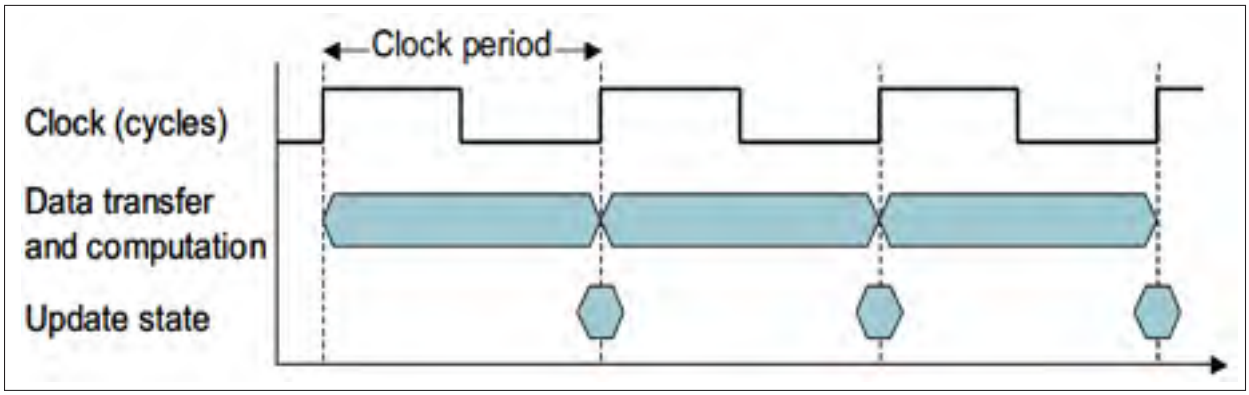

Figure 3.2 opération de matériel numérique régie par un débit constant de l'horloge

Nous pouvons appuyer notre choix des ces métriques par une revue littérature dans ce domaine qui a été présentée dans la section 2.3 "travaux connexes", soit les métriques utilisées pour analyser la performance dans le Cloud Computing en général.

De façon plus précise et pointue, le temps d'exécution et CPU ont été définis étant deux métriques très importantes et suffisantes, décrivant qualitativement la performance du Cloud

Middleware.

En résumé, quand nous abordons le sujet du choix des métriques à collecter, plusieurs considérations sont prises en vu : nous devrons sélectionner les métriques basées sur la disponibilité (availability), la précision (accuracy), le coût (cost) et la rapidité (timeliness) avec :

– disponibilité : Les données doivent être collectées facilement. Autrement dit, elles doivent être faciles à mesurer et linéaires dans ses changements de valeur par le même rapport ;
– précision : Les données colletées doivent être fiables et crédibles, avec calculs exacts et cohérents dans le temps. Cela veut dire que la métrique à répétabilité doit donner la même valeur, à chaque fois que nous faisons la même mesure ;
– rapidité : Les données doivent être suffisamment rapides pour évaluer la performance ;
– coût : Il faut avoir un coût minimal pour collecter les données. De plus, il faut avoir suffisamment de ressources disponibles pour la collecte de ces données.

Nous résumons cette partie, en soulignant les points suivants : premièrement, les métriques à superviser sont la charge de travail du CPU et le temps d'exécution pour compléter un scénario ; deuxièmement, le scénario choisi est la création et la suppression d'un lot de machines virtuelles distribuées sur un ensemble de locataires et d'utilisateurs. Nous résumons le processus de collecte des données dans la figure suivante :

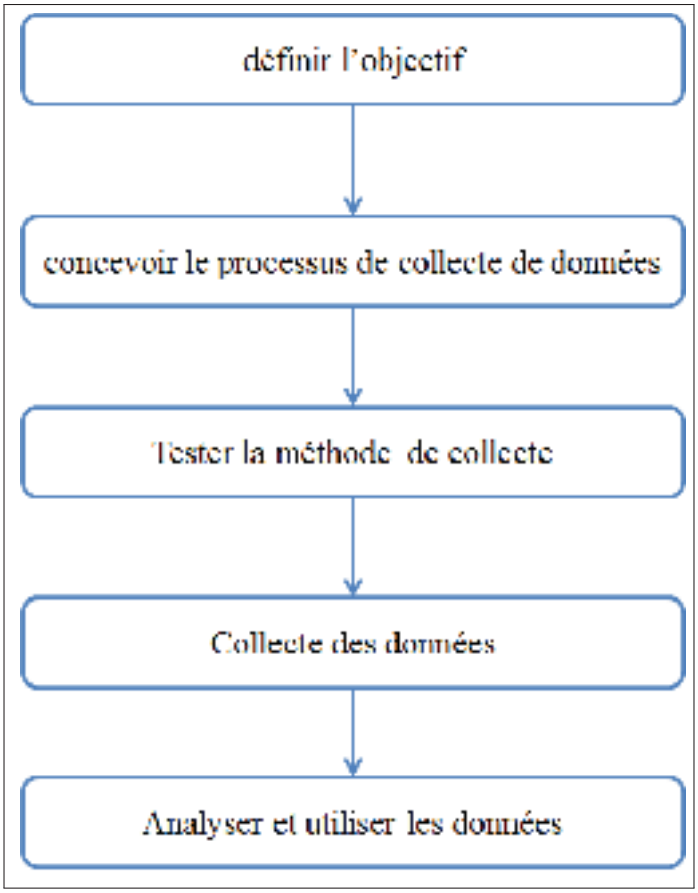

Figure 3.3 Processus de collecte de données pour les mesures de performance

Nous mettons en évidence que nous pouvons appliquer la même idée de la supervision de la consommation des ressources mais en changeant le scénario. Par exemple, au lieu de faire la création et la suppression des instances, nous nous focalisons sur la performance du Cloud Middleware lors de la migration entre les nœuds compute existants du Cloud et nous choisissons quand et comment les migrer. Ou bien, nous utilisons une couche réseau complexe qui nécessite d'isoler les locataires et de faire passer l'instance sur des parts-feu avant de la créer ; puis nous calculons et comparons les résultats afin de comprendre la performance du Cloud dans ce genre de situation, etc.

En revanche, pour ces deux exemples que j'ai présentés, je les classe dans deux catégories de deux domaines différents et qui sont interconnectés à la fois ; à savoir le domaine de la

gestion des ceenter de données et le domaine de la sécurité qui cible ces ceenter de données. Comme nous constat, il existe plusieurs scénarios et plusieurs manières pour définir la problématique et pour donner une approche afin de résoudre le problème.

Pour clôturer cette section, je réponds à une question que le lecteur peut se poser au cours de sa lecture "*Pourquoi j'ai choisi la création et la suppression de l'instance, étant mon seul et unique scénario étudié ? Ainsi que, pourquoi me limiter aux trois facteurs et ne pas prendre en considération les types de systèmes d'exploitation à avoir une variété de types des instances à créer ? Puis observer l'impact de ce genre de scénario sur la performance de mon Cloud Middleware.*"

L'instance est le seul élément qui interagit avec toutes les couches du Cloud et plus particulièrement, le Cloud Middleware. Par conséquent, nous devons nous assurer que l'instance fonctionne correctement sans aucun problème, puis nous pouvons associer à cette instance des charges de travail (tâches) afin d'être exécutées proprement dans le Cloud.

Avant d'investiguer si mon Cloud midleware peut supporter la migration, gérer, assurer la sécurité et/ou l'isolation de mes instances. Est-ce que ce Cloud Middleware peut créer une série d'instances proprement dites et pouvant être supprimées par la suite sans aucun problème. Vu que l'approvisionnement d'une machine virtuelle est complexe et doit, nécessairement, passer par une série d'étapes avant d'être prête à l'utilisation ; nous citons l'identification de la requête, choix du nœud de l'hôte, en nous basant sur le mécanisme d'ordonnancer, de saisir et d'allouer des caractéristiques de l'instance comme CPU/RAM/OS, communication avec l'hyperviseur pour héberger l'instance etc. (voyons plus de détails dans le quatrième chapitre).

Par ailleurs, pour répondre à la deuxième question, je me suis limité aux trois facteurs, car je veux savoir est-ce qu'il existe une relation entre ces derniers. De plus, j'ai négligé les autres facteurs comme le choix des divers VCPU, RAM et système d'exploitation qui seront associés à mes instances. Tout simplement, ces facteurs négligés sont, entre autre, des ressources liées

à mon instance qui seront elles aussi liées à mes ressources matérielles. Ce qui revient à dire que si je veux une capacité à une instance je dois m'assurer d'abord si j'ai déjà suffisamment de ressources physiques. Dans le cas de mon étude, j'envisage de créer le maximum possible d'instances, afin d'observer comment Cloud Middleware (trois facteurs) va se comporter face cette charge. Ainsi, mon intérêt n'est pas focalisé sur la capacité de l'instance à créer.

3.2 Méthode de la collecte de données

Il y a plusieurs raisons pour lesquelles la collecte de données sur des serveurs, des applications est utilisée. La collecte et l'organisation des données peuvent donner certaine crédibilité en prise de décision, concernant l'extension (scaling), le dépannage (troubleshooting) afin de changer la mise à jour ou d'enlever des composants ou modifier les fichiers de configurations.

Dans ce projet, notre méthode de collecte de données vise deux catégories de données, à savoir : des données reliées à l'infrastructure et des données reliées au Cloud Middleware. Ces deux processus de collecte se font en parallèle. Ces deux processus ont pour but de trouver la relation entre les facteurs du Cloud Middleware et de calculer l'impact de ces facteurs sur la performance de l'infrastructure utilisée. Dans ce qui suit, nous allons présenter quelques outils reliés à la collecte de données de l'infrastructure comme la consommation du CPU et le temps d'exécution ; ce qui répondra à la deuxième question de la problématique.

3.2.1 Méthode liée à l'infrastructure

En réalité, il existe une variété d'outils qui peuvent être utilisés pour superviser les métriques ciblées dans les systèmes étudiés. Très souvent, ils occupent une certaine partie du processus. Ce genre d'outils peut être ajouté ou combiné (plug-ins) pour créer un système de collecte, d'enregistrement et d'affichage des résultats.

La raison primordiale, pour laquelle nous utilisons un outil de collecte de donnée, est en fait assez simple : plus nous avons de données et plus nous serons en mesure de comprendre ce qui

se passe à un moment donné. Cela nous donne la capacité remarquable, à la décision, avec les données brutes et nous permet de voir à l'avance, si un changement susceptible est ciblé au bon composant. Le suivi des statistiques nous fournit une source supplémentaire d'informations qui peuvent ne pas être présentes dans les logs des applications.

3.2.1.1 Collectd

Une des façons les plus faciles à recueillir des informations détaillées sur un serveur est un démon nommé collecte.

Collecte peut rassembler des statistiques sur de nombreux composants différents d'un environnement de serveur. Il permet de suivre facilement les paramètres communs, comme l'utilisation de la mémoire, la charge CPU, le trafic réseau, etc. Cela permet de corréler facilement les événements avec l'état du système cible.

Au-delà de la collecte d'informations de système standard, collecte dispose également d'un système de plug-in qui étend ses fonctionnalités. Cela signifie que nous pouvons facilement collecter des informations du logiciel comme Apache, Nginx, iptables, memcache, MySQL, PostgreSQL, OpenVPN, et beaucoup plus.

Collecte fournit un moyen simple d'obtenir des données à partir d'applications pré configurées et des services communs sur les serveurs. Cela doit être utilisé pour suivre le comportement de l'infrastructure et les services qui sont exécutés sur ce dernier. (Voir l'annexe).

3.2.1.2 StatsD

StatsD est un démon très simple qui peut être utilisé pour envoyer des données à d'autres Graphite (c'est une application Web qui permet de visualiser et de concevoir des graphiques qui tracent les données collectées par Stats). L'avantage de cette approche est qu'elle devient triviale à construire dans le suivi de stat aux applications et aux systèmes que nous créons.

StatsD fonctionne en écoute sur une interface pour les paquets UDP simples qui représentent un point de données unique. Cela signifie qu'il peut accepter une grande quantité d'informations d'une manière sans connexion. Il peut alors agréger les valeurs qu'il reçoit et les transmettre à Graphite.

Ce système nous permet d'envoyer les statistiques en grandes quantités sans nous soucier de la croissance de latence de l'application. Le service StatsD permettra de recueillir toutes les données, de les agréger, puis les envoyer sous forme d'un résumé de points de données pour graphite à chaque laps de temps.

En raison de ces avantages, il est en fait un bon intermédiaire pour tout type de données envoyées à Graphite. Mais, le principal moyen que nous pouvons tirer à partir de cet outil est de pouvoir surveiller nos propres applications et les outils que nous créons.

StatsD est parfait pour cela, car il est un démon d'usage général qui accepte le trafic UDP. Il y a beaucoup de différentes bibliothèques, côté client dans divers langages de programmation qui peuvent envoyer des données directement à une instance StatsD. Cela signifie que les applications, que nous construisons, peuvent facilement envoyer des données à être suivies.

3.2.1.3 Nagios

Nagios Core ^{TM} est un système Open Source, l'application de surveillance réseau. Il surveille les hôtes et services que nous spécifions, pour nous avertir quand il y a un changement critique.

Le core de Nagios a été initialement conçu pour fonctionner sous Linux, il supporte la plupart des autres distributions Unix.

Parmi les nombreuses fonctionnalités de Nagios, les bases comprennent :

- la surveillance des services réseau (SMTP, POP3, HTTP, NNTP, PING, etc.) ;
- le Monitoring des ressources (charge processeur, l'utilisation du disque, etc.) ;
- le design des simples plug-ins qui permettent aux utilisateurs de développer facilement leurs propres contrôles de service ;
- les contrôles de service parallélisés ;
- l'interface Web option pour visualiser l'état du réseau, la notification et le problème histoire actuelle, le fichier journal, etc.

En effet, il existe d'autres outils qui ont plus au moins les mêmes fonctionnalités des outils cités ci-dessus et sont plus ou moins populaires. Par ailleurs, notre choix de l'outil doit suivre la contrainte suivante ; il faut utiliser un outil qui collecte les données à moindre coût. Autrement dit, il ne faut pas que l'outil utilisé ait un impact sur la performance du système, car en général tous les outils de collecte prennent une partie du processus lors de son exécution.

Durant mes recherches, je peux dire que les deux outils (logiciels) les plus connus, utilisés et gratuits (open-source) sont Nagios et Collectd.

Concernant l'outil Nagios, il est recommandé, seulement si, notre vision est de superviser un large et complexe réseau qui se constitue de plusieurs commutateurs (switchs) et routeurs (routers) ; premièrement, nous pouvons comprendre que Nagios est plus dédié au monitoring des réseaux et deuxièmement, nous constatons, implicitement, que Nagios lui aussi a une bonne part des ressources physiques pour qu'il puise être exécuté, afin de collecter les données (métriques) souhaitées. D'où, Nagios n'est pas le bon candidat pour collecter nos métriques vu que l'outil que nous voulons utiliser, doit suivre deux contraintes : une collecte précise et à moindre coût (impact négligeable).

En contrepartie, nous avons Collectd, mis à part de ce qui a été présenté, nous rajoutons ici

que Collectd inclut des optimisations et des fonctionnalités, pour gérer des centaines de milliers d'ensembles de données. Il est livré avec plus de 90 plug-ins qui varient entre les cas standard allant jusqu'à des sujets très spécialisés et avancés. Il peut être extensible de nombreuse façons. Dernier point, mais non des moindres : collecte est activement développé et soutenu et bien documenté. Une liste plus complète des fonctionnalités est disponible. Et surtout Collectd est une application écrite en C et est conçu pour fonctionner comme un démon, ce qui en fait une application *faible en surcharge (overhead)* qui est capable de consigner les informations à de courts intervalles, *sans impact significatif sur le système*.

De ce fait, nous pouvons dire que le choix du Collectd est une candidate potentielle pour la collecte des données. Mais, puisque nous travaillons dans un environnement LINUX, les outils cités auparavant utilisent tous des fonctionnalités déjà existantes dans le noyau du LINUX. Aussi, nous nous sommes intéressés uniquement aux deux métriques qui sont la consommation du CPU et le temps d'exécution, nous avons opté à écrire notre propre script pour la collecte des données souhaitées comme présenté dans la synoptique suivante.

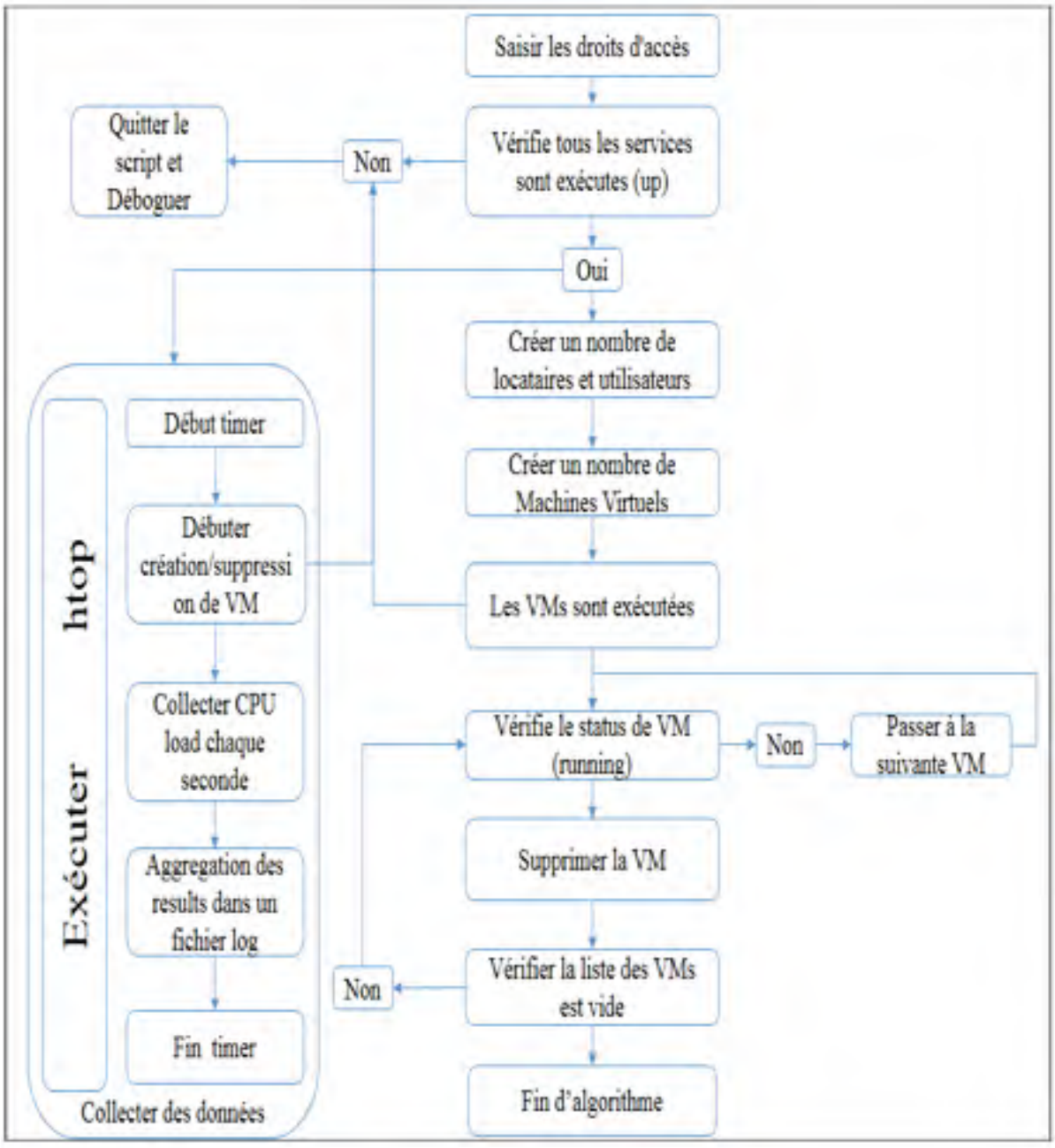

Figure 3.4 Algorithme de la création et suppression des instances et la collecte de données

En effet, le script est une combinaison de deux fonctionnalités qui roule à la fois en parallèle. Le premier est le scénario de la création et de la suppression (Add and Drop) des machines virtuelles et le deuxième est la collecte de la charge de CPU et du temps d'exécution qui est causée par la première fonctionnalité. Nous présentons la démarche du processus de chaque fonctionnalité.

Scénario de la *création et suppression* (Add and Drop) :

1 - saisir les droits d'accès au cloud privé ;

2 - vérifier que tous les services du Cloud sont connectés, si un des services n'est pas connecté, nous quittons et nous déboguons pour trouver l'erreur ;

3 - sinon, nous commençons à créer d'abord un ensemble de locataires et d'utilisateurs, les machines virtuelles seront accessibles à ces utilisateurs qui sont associés à ces locataires ;

4 - puis, nous débutons la création des machines virtuelles en suivant le plan d'expérience et le processus de la collecte des données (voir figure 3.4) ;

5 - si les machines virtuelles ne sont pas créées correctement, nous quittons et nous déboguons pour trouver l'erreur ;

6 - avant de commencer le processus de suppression des machines virtuelles, il faut vérifier que le statut de ces machines est en marche (running), sinon nous passons à la machine suivante ;

7 - si la machine virtuelle est au statut running, nous la supprimons ;

8 - vérifier s'il reste d'autre machine virtuelle dans le Cloud ;

9 - si oui revenir à l'étape ;

10 - sinon quitter l'algorithme.

Deuxième fonctionnalité *la collecte des données* :

1 - exécute la commande htop tout au long du processus de la collecte afin de tirer la consommation du CPU ;

2 - lancer le minuteur (timer) ;

3 - débuter le scénario de la création et la suppression (Add and Drop) des machines virtuelles ;

4 - extraire la variation du CPU chaque seconde ;

5 - agréger les valeurs collectées puis les enregistrer dans un fichier log ;

6 - arrêter le minuteur à la fin de l'algorithme ;

7 - afficher le temps pris pour réaliser la première fonctionnalité.

3.2.2 Choix du plan expérimental

Cette partie traitera la troisième question de la problématique. Avant de procéder au choix du plan d'expérience le plus adéquat dans notre recherche, nous rappelons que les plans d'expériences s'appliquent pour répondre aux exigences de deux grandes catégories de problèmes. La première catégorie de problèmes, auxquels les plans d'expériences peuvent être utilisés pour faciliter le processus des expériences durant la phase des tests, est dite criblage ou tamisage (screening) faisant partie de la famille des plans factoriels fractionnaires. Ce genre d'approche consiste à construire une idée sur l'effet des facteurs (paramètres) qui entre en jeux durant les expériences.

Autrement dit, nous utilisons le tamisage afin de raffiner le nombre des facteurs, dans le cas où nous avons assez de facteurs, et nous n'avons pas une idée claire au préalable sur leurs comportements au sein du système. Ceci se fait par le biais d'une combinaison bipolaire des facteurs retenus, dans le but d'extraire les facteurs les plus influents ainsi que leurs interactions existantes.

Généralement, cette technique est réalisée par le biais des plans factoriels complets à deux niveaux ou par les plans factoriels fractionnaires. Par ailleurs, en ce qui concerne la deuxième

catégorie des plans d'expérience, nous les nommons "Plans de surface de réponse". Entre autre, il existe plusieurs plans qui définissent les plans de surface de réponse (plans Box-Behnken, plans composites centrés et autre). Cette catégorie vise à déterminer d'une façon quantitative les variations de la réponse, vis-à-vis des facteurs d'influence significative.

Pour présenter la technique du sceerning, il existe plusieurs travaux qui sont faits auparavant et sont discutés dans la littérature. Nous illustrons un de ces exemples, en (Srinivasaiah et Bhat, 2005) l'auteur a utilisé un plan à 28 essais pour identifier et mettre en évidence 10 facteurs potentiels qui affectent positivement parmi un ensemble de 21 facteurs sur le comportement du CMOS (Complementary Metal Oxide Semi-conductor). Cela limite le nombre de facteurs à étudier, ce qui implique moins de nombres d'essais.

3.2.2.1 Définition de l'objectif et la réponse

En se basant sur ce qui précède cette section, nous avons mentionné que la construction du plan d'expérience dépend de façon étroite de l'objectif visé. Cela nous guide à définir l'objectif de l'étude ainsi que la réponse observée de façon précise.

Dans la section "les métriques à superviser", nous avons défini les facteurs potentiels envers la création d'une machine virtuelle, aussi nous avons expliqué pour quelle raison nous avons choisi le plan de tamisage comme étant notre plan d'expérience adopté. À la lumière de ces données, il nous reste un seul élément à définir.

Pour compléter l'image globale de l'expérience, nous devons spécifier la valeur résiduelle des effets interaction entre les facteurs cités auparavant (cette grandeur est dite la réponse dans la terminologie des plans d'expériences). De ce fait, la réponse sera sous forme d'un taux de pourcentage de réussite de chaque simulation effectuée. Les modifications, dans les paramètres des facteurs, engendrent une nouvelle observation dont une variation sur la réponse.

Dans la synoptique ci-dessous, 3.5 nous montrons la méthode de la collecte des données liée au Cloud Middleware. À l'aide de cette méthode, du script pour calculer la consommation

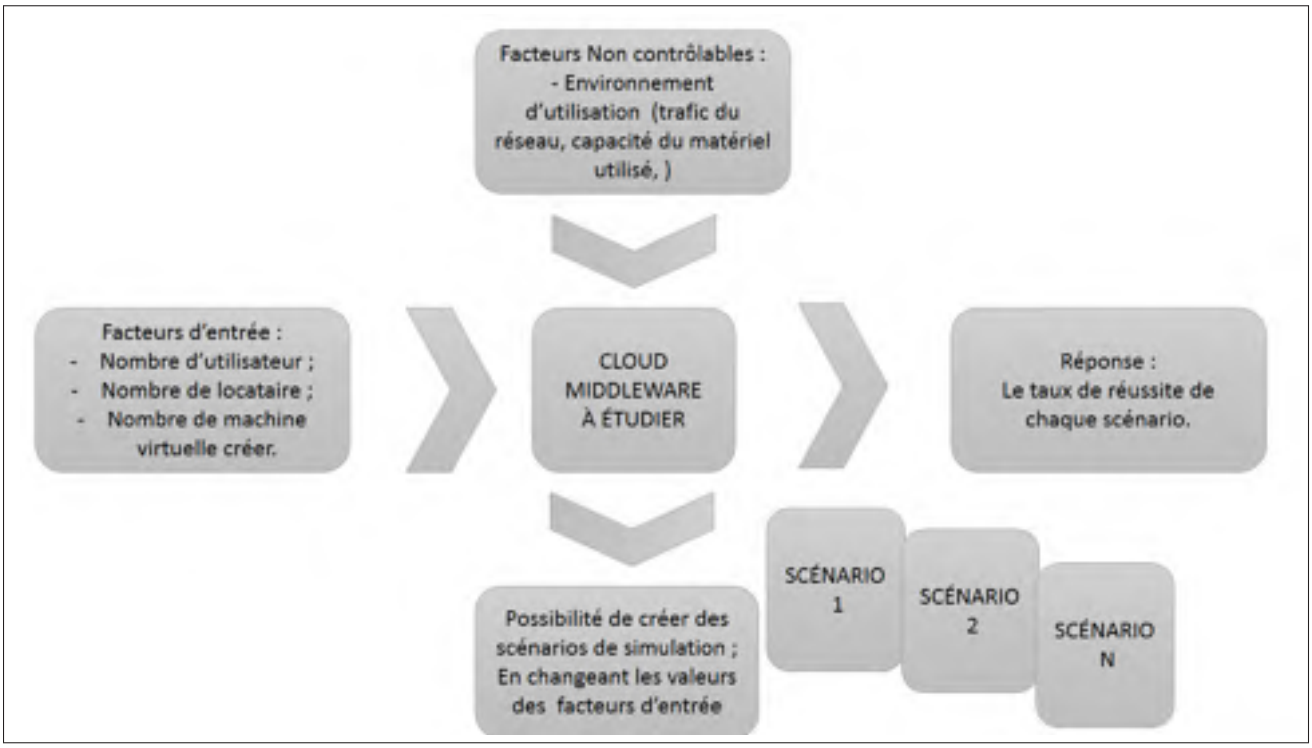

Figure 3.5 Méthode de la collecte des données liée au cloud middleware

du CPU et du temps d'exécution durant les simulations, nous pourrons à la fois déterminer la relation entre les facteurs, extraire le facteur le plus significatif, les niveaux de saturation et préciser l'influence de la couche physique sur la performance du Cloud Middleware. Cela a pour but de répondre à la quatrième question de la problématique qui est le calcul de l'impact des ressources physiques sur la performance du Cloud Middleware.

Pour résumer cette section, après le déploiement du Cloud Middleware, notre la méthodologie proposée consiste à construire un modèle de simulation pour le scénario opté et discuté dans la sous-section précédente, de modéliser les comportements à l'aide de plans d'expériences, d'optimiser le nombre de facteurs (paramètres) influençables (s'ils existent) par l'entremise des plans factoriels (tamisage) et de calculer l'impact de ces simulations sur l'infrastructure

du Cloud Middleware. Une vue globale de la méthodologie est donnée dans le diagramme 3.6 suivant :

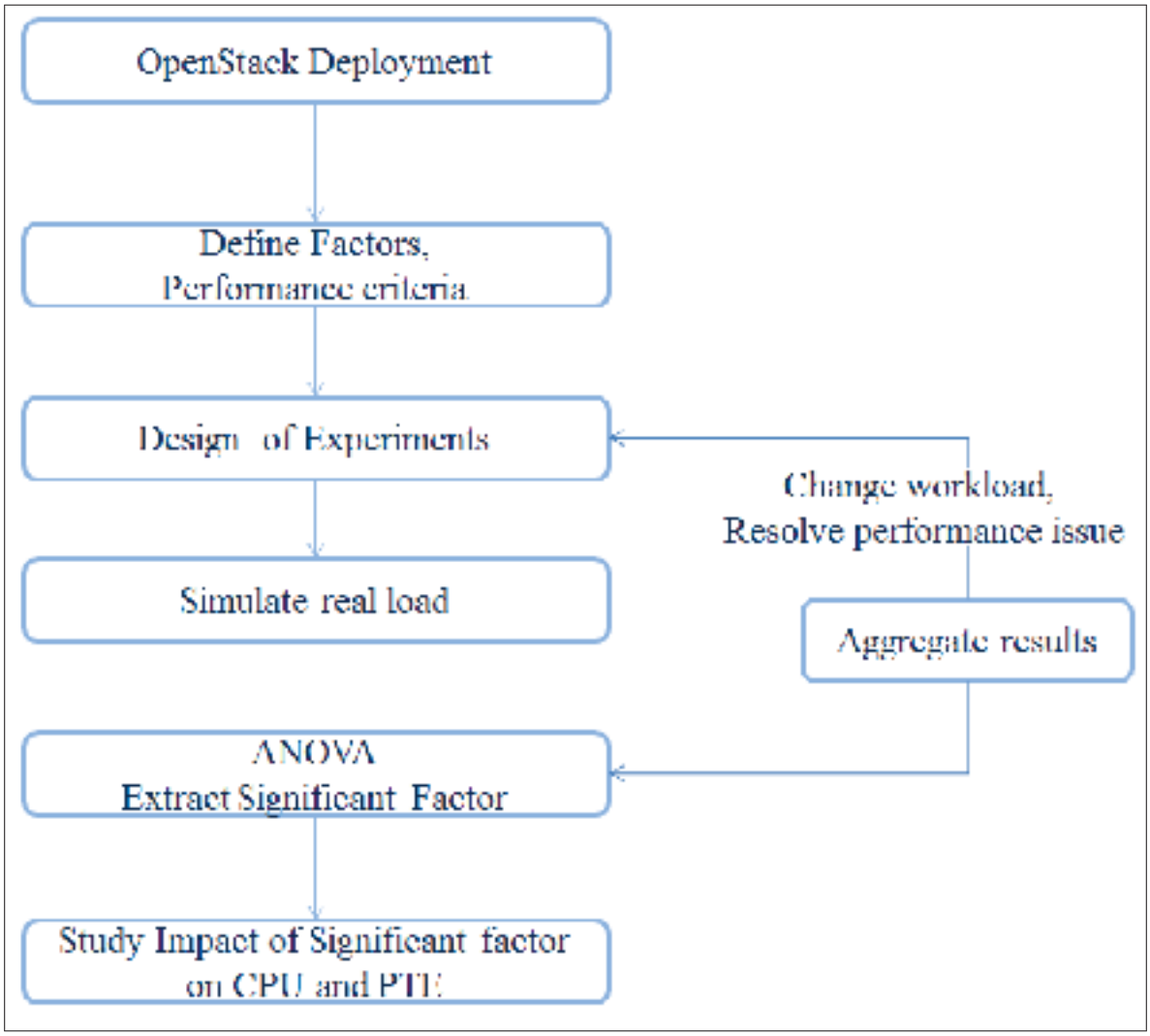

Figure 3.6 l'organigramme de la méthodologie de travail proposée

1 - déployer OpenStack sur une architecture distribuée ;

2 - définir les métriques à superviser ;

3 - définir le plan d'expérience choix ;

4 - simuler le scénario de création et la suppression d'un groupe d'instances avec différente charge de travail ;

5 - collecter les données puis agréger ces résultats ;

6 - extraire le facteur le plus significatif et formuler l'analyse de la variance ;

7 - évaluer et étudier l'impact du facteur signifiant sur les ressources matérielles des serveurs.

3.3 Conlusion

En guise de conclusion à ce chapitre de la méthodologie de travail, nous avons déterminé les métriques décrivant la performance du Cloud Middleware, soit locataire, utilisateur, instance. Puis, nous avons défini une liste précise des métriques à superviser en phase des tests soit le temps d'exécution et la charge du CPU afin de comprendre l'impact de ces derniers sur l'infrastructure. Et nous avons défini une méthodologie de tests en nous basant sur les plans d'expériences afin de structurer nos tests.

CHAPITRE 4

EXPÉRIMENTATIONS ET RÉSULTATS

4.1 Introduction

À la lumière des problématiques et objectifs décrits dans les sections 1.2 et 1.3, et la démarche proposée dans le chapitre précédent ; ce présent chapitre portera sur les expérimentations réalisées, ainsi que les résultats obtenus afin de consolider avec ce qui a été élaboré dans les sections antérieures. Nous avons conçu et implémenté un modèle générique pour profiler et évaluer la consommation des ressources matérielles des équipements utilisés.

Dans ce qui suit, nous allons présenter l'environnement expérimental, suivre les différentes étapes de la méthodologie pour atteindre les objectifs du mémoire de recherche et exposer les scénarios exécutés sur le banc d'essai et leurs résultats. Des discussions de ces derniers ainsi que des synthèses de performances seront retenues à la fin de ce chapitre.

4.2 Environnement expérimental

4.2.1 Serveur de banc d'essai

Ericsson Server Platform est plus robuste et tolérant aux pannes que toute technologie de serveur comparable. Il est extrêmement fiable avec une capacité linéairement évolutive, et les caractéristiques en temps réel, ce qui signifie que la transmission a lieu dans un délai minimal et contrôlé. Le logiciel de télécommunications de qualité est activé par TelORB clusterware, qui tourne au-dessus de DICOS et de Linux. Le logiciel intègre la toute dernière dans le système de signalisation no. 7 (SS7) et intégré dans la gestion de nœud, avec le soutien de nombreux protocoles nécessaires à l'interopérabilité entre la plate-forme Télécom Server et des systèmes de soutien des opérations (OSS) (Ericsson, 2015) par ces points forts, nous trouvons :

– flexibilité - Blade générique qui prend en charge toutes les applications basées X.86 y compris ceux de tiers ;
– NEBS 3 compatible - cela est essentiel pour fournir haut dans la performance de service requise dans les réseaux de télécommunications, mais aussi de réduire les coûts de maintenance et de réparation ;
– nuage prêt - prêt à fonctionner dans un environnement de Cloud Computing avec des fonctions de contrôle du réseau et de la charge utile virtualisés.

4.2.2 Déploiement du banc d'essai

Notre banc d'essai est réalisé, voir figure 4.2 qui est une illustration plus détaillée des composantes de la figure dans la synoptique 4.1. Ce banc d'essai déployé suit la topologie de réseau en étoile. Cette dernière se constitue de trois serveurs (Ericsson blade GEP-3 blade), dont les caractéristiques techniques sont présentées dans le Tableau 4.1, chacun est attaché au commutateur avec une liaison directe et d'un débit de 1 GB.

Nous notons, ici, que la technologie de virtualisation utilisée est KVM, voir l'annexe I pour plus de détails.

Tableau 4.1 Caractéristiques des serveurs hébergeurs du banc d'essai

Serveur	**Système d'exploitation**	**CPU**	**Mémoire**	**Contenu**
Serveur 1	Ubuntu Server 12.04 x64 TLS	12CPU x 2GHz	24G	Controller /Compute
Serveur 2	Ubuntu Server 12.04 x64 TLS	12CPU x 2GHz	24G	Compute
Serveur 3	Ubuntu Server 12.04 x64 TLS	12CPU x 2GHz	24G	Compute

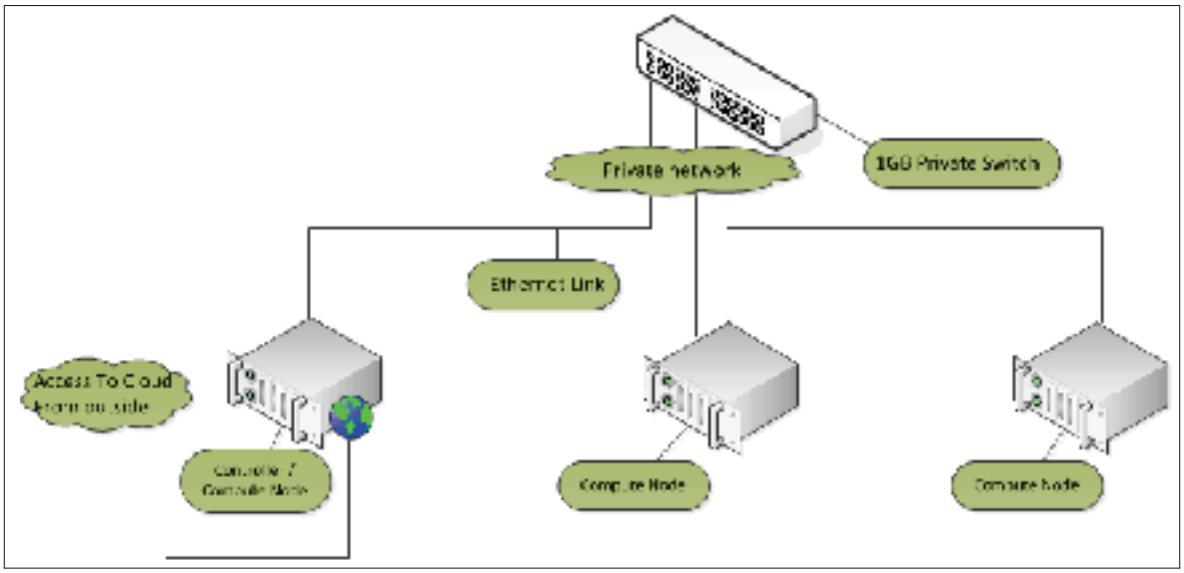

Figure 4.1 Architecture abstraite de la topologie

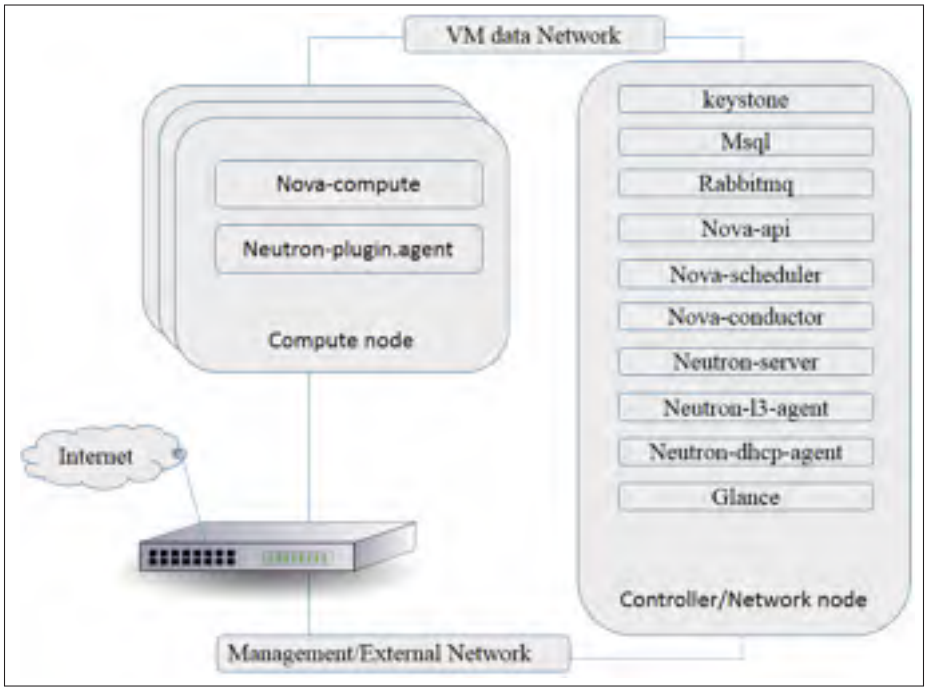

Figure 4.2 Les différents composants des services d'OpenStack utilisés

4.3 Déploiement de OpenStack

Dans cette recherche, nous nous concentrons sur les caractéristiques qui ont l'impact le plus important sur la performance de OpenStack (notons ici que nous avons utilisé la version Ice-House d'OpenStack).

En d'autres termes, le but de notre expérimentation est de déterminer la limitation ou goulot d'étranglement OpenStack, en créant des instances simultanées dans la limitation de la capacité de matériel, puis savoir s'il existe une corrélation entre le nombre d'instances créées, le nombre de locataires et le nombre d'utilisateurs par locataire, la performance de Cloud en termes du temps nécessaire pour la création des instances et la capacité de CPU utilisé comme expliqué dans chapitre "Méthodologie de travail". Pour observer comment OpenStack gère les demandes simultanées, nous mettons en place un environnement de matériel sous-jacent comme le montre la Figure 4.1.

Nous avons mentionné, dans le chapitre précédent, notre scénario principal qui est simultanément, la création et la suppression des instances par la création d'un certain nombre d'instances en parallèle, afin d'étirer notre pile (Stack) au maximum et d'observer la réponse d'OpenStack comment réagira-t-il face à ces requêtes.

Aussi, nous avons mis en évidence, dans le chapitre méthodologie de travail unifié, les instances utilisées ainsi que l'importance de ce point afin d'obtenir des résultats objectifs. Sur ce, nous configurons nos instances avec une exigence minimale, en utilisant :

- image : "cirros-0.3.2-x86_64-uec" (Cirros nuage image) ;
- type (flavor) : m1.nano ;
- CPU : un seul virtuel CPU (VCPU) ;
- RAM : 64 Mo.

4.3.1 Provisionnement d'une instance

Jusqu'à présent nous n'avons pas encore expliqué le mécanisme de la création d'une machine virtuelle à l'aide d'OpenStack. La partie suivante met en évidence les étapes nécessaires pour approvisionner une instance. Ces étapes nous aideront à comprendre le mécanisme de la création d'une instance. Sur ce, cette démarche sera utile pour la suite dans la partie des discussions des résultats. Aussi, les mesures prises pour mettre en place une instance, comme l'illustre

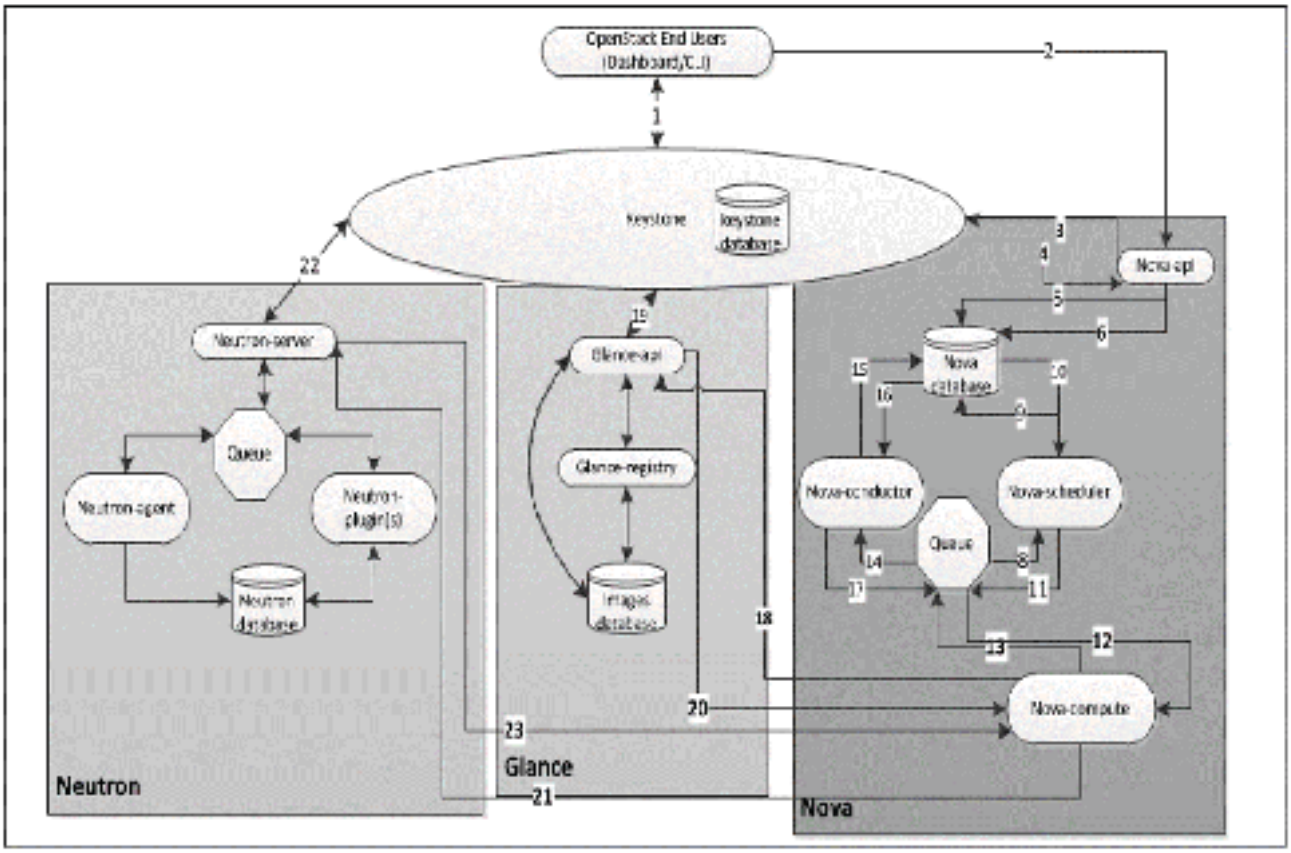

Figure 4.3 les étapes nécessaires pour le provisionnement d'une machine virtuelle

le processus d'approvisionnement, comprennent de nombreuses étapes, numérotées de 1 à 23 afin de construire une instance (voir Figure 4.3). Pendant le processus, le nœud du contrôleur envoie différentes demandes dans un ordre chronologique, commençant par la vérification

des identifiants, terminant par la création des instances sur le nœud compute et passant par la communication entre les composants (nova, Glance et Neutron) via le protocole de communication/messagerie RabbitMQ (voir figure 4.3).

Dans ce qui suit, nous allons décrire et expliquer chaque transaction (numérotées de 1 à 24) dans Figure 4.3 ci-dessous :

étape 1 à 2 : la vérification des identifiants : à savoir le tenant et l'utilisateur qui existe dans le tenant, de plus vérifier le quota pour savoir si ce dernier a le droit de créer des instances, puis Keystone d'identifier et de générer les informations puis renvoyer haut-token qui sera utilisé pour l'envoi de la demande d'autres composants via l'appel de REST ;

étape 3 : demande de créer une nouvelle instance via le tableau de bord ou par commande de ligne REST API aux nova-api,

étape 4 : Nova-api reçoit la demande et l'envoie pour valider auth-jeton (auth-token) et pour accéder à Keystone ;

étape 5 : Keystone valide le jeton et envoie la réponse avec les rôles et les autorisations ;

étape 6 : Nova-api interagit avec la base de données de nova (nova-database) ;

étape 7 : Créer une entrée dans la base de données initiale pour allouer une nouvelle instance ;

étape 8 : Nova-api envoie la demande à nova-scheduler via de rpc.call pour obtenir l'entrée d'instance avec l'ID d'hôte spécifié ;

étape 9 : nova-scheduler choisit la demande de la file d'attente ;

étape 10 : nova-scheduler interagit avec nova-database pour trouver l'hôte approprié ;

étape 11 : Renvoie l'entrée de l'instance mise à jour avec l'ID d'hôte approprié ;

étape 12 : nova-scheduler envoie la demande de rpc.cast à nova-compute pour lancer l'instance sur l'hôte approprié ;

étape 13 : Nova-compute prend la demande de la file d'attente ;

étape 14 : Nova-compute envoie la demande de rpc.call à la Nova-conducteur pour aller chercher les informations d'instance telles que l'ID de l'hôte et la saveur (Ram, CPU, disque) ;

étape 15 : Nova-conducteur reprend la demande de la file d'attente ;

étape 16 : Nova-conducteur interagit avec la Nova-base ;

étape 17 : Retourner les informations d'instance ;

étape 18 : Nova-compute prend les informations d'instance de la file d'attente ;

étape 19 : Nova-compute fait l'appel REST en passant auth-jeton aux glance-api pour obtenir l'image URI par Image ID de glance et de télécharger l'image de stockage d'images ;

étape 20 : glance-api valide l'auth-token avec keystone ;

étape 21 : Nova-compute obtient les méta-données d'image ;

étape 22 : Nova-compute fait l'appel REST-call en passant auth-token au API du réseau pour allouer et configurer le réseau de telle sorte que la nouvelle instance obtienne une adresse IP ;

étape 23 : Neutron serveur valide l'auth-token avec keystone ;

étape 24 : Nova-compute obtient les informations du réseau ;

étape 25 : Nova-compute fait l'appel REST en passant auth-token à API Volume pour attacher le volume à l'instance ;

étape 26 : Cinder-api valide l'auth-token avec keystone ;

étape 27 : Nova-compute obtient l'information de stockage de bloc ;

étape 28 : Nova-compute génère les données au l'hyperviseur pour l'exécuter via libvirt ou API.

Ces étapes nous aideront à comprendre le mécanisme de la création d'une instance qui est la base de notre scénario. Aussi, cette démarche est nécessaire et utile, pour la suite dans la partie des discussions des résultats.

4.3.2 Plan factoriel complet

Dans les statistiques, une expérience factorielle complète est une expérience dont la conception se compose de deux ou plusieurs facteurs, chacun avec des valeurs discrètes possibles aux

«niveaux», et dont les unités expérimentales prennent toutes les combinaisons possibles de ces niveaux à travers tous ces facteurs. Une telle expérience permet au chercheur d'étudier l'effet de chaque facteur sur la variable de réponse, ainsi que les effets de facteurs sur les interactions avec la variable de réponse.

Notre plan d'expérience est basé sur l'utilisation du plan de conception de dépistage (tamisage) pour évaluer et déterminer l'importance de chaque variable indépendante. En effet, nous avons défini les variables indépendantes dans la section 3.1.1 comme suit :

– nombre d'utilisateurs (user) : chaque instance créée est utilisée par un ou plusieurs utilisateur(s) ; voir figure 4.3 étapes 1-5 ;
– nombre de projet (tenant) : dans chaque projet, nous trouvons une ou plusieurs instances qui sont accédées par un ou plusieurs utilisateur(s) ; voir figure 4.3 étapes 1-5 ;
– nombre d'instances : nombre d'instances que nous pouvons créer. (pour chaque instance le processus se répète voir figure 4.3 étapes 5-28).

L'objectif de cette méthodologie adoptée tourne autour de deux points principaux :

– trouver les relations avec ces paramètres, ainsi que leurs interactions qui ont un effet significatif sur la variable de réponse ;
– optimiser le nombre des paramètres possibles.

Pour résumer, nous nous sommes préoccupés de l'identification qui est facteur à effet important sur la variable dépendante (taux de réussite %) sur OpenStack. Ce type de plan factoriel complet est souvent utilisé et recommandé comme modèle attribué pour un certain nombre de facteurs moins de (< 5), cela donne des résultats plus précis, car chaque interaction est estimée séparément (MONTGOMERY, 2001e). Trois réplications ont été effectuées pour chaque combinaison de facteurs, et donc 24 $(2^3 * 3)$, simulations ont été exécutées. À partir de quelques expériences de simulation préliminaires, les plages du domaine expérimental ont été établies, elles sont présentées dans le tableau suivant : Nous avons utilisé **Statgraphics** comme outil

Tableau 4.2 Niveaux des variables indépendantes

Facteur	Niveau Inférieur	Niveau Supérieur	Description
F_1	100	500	Nombre d'instances
F_2	1	80	Nombre de projets
F_3	10	60	Nombre d'utilisateurs

pour réaliser nos tests afin d'extraire le(s) facteur(s) le plus signifiant(s). Entre autre, Statgraphics est un progiciel statistique qui permet d'effectuer et d'expliquer les fonctions statistiques de base avancées. (Voir annexe pour plus de détail).

Notons ici que le *taux de réussite* est présenté dans le chapitre précédent ; se référer au :

– Nombre de demandes de suppression qui ont été demandées mais non réalisées ;
– ET, se référer aux cas où les demandes de suppression ne peuvent pas être supprimées, car elles ne sont pas créées en premier lieu.

4.4 Résultats Et interprétation

Suivant le plan d'expérience et l'algorithme décrit dans le chapitre précédent, nous avons effectué une série de tests.

Les tableaux représentent un résumé des différents scénarios de la création et de la suppression des machines virtuelles, répétés cinq fois, afin de valider les résultats obtenus (la collecte des métriques). Nous rappelons que nous avons défini quatre critères (Disponibilité, Précision, Rapidité et Coût) voir section 3.1.1.

– nova.boot_server : se réfère à la création de la machine virtuelle ;
– nova.delete_server : se réfère à la suppression des machines virtuelles ;
– min, Avg, max(sec) : se réfèrent au temps pris pour terminer le scénario ;
– success : se réfèrent au taux de réussite de chaque scénario ;

– before, after : se réfèrent au cas de changement (déboguer) dans la configuration afin d'améliorer la performance du OpenStack s'il y a eu lieu.

Nous expliquons la baisse du taux de réussite dans le tableau 4.3 (success/before) avant d'effectuer un changement dans la configuration d'OpenStack, par le fait qu'OpenStack installe ces composants avec des paramètres par défaut. Il attribue à chaque projet (tenant) un nombre limité d'instances (voir 10 instances au maximum). Pour contourner ce problème, nous avons deux approches à suivre. La première approche est de réguler pour chaque projet (tenant) un nombre d'instances plus que 10, c'est à dire à chaque fois nous faisons un changement, dépendant des entrées du scénario (nombre d'instance, utilisateur, locataire). La deuxième approche est de définir dès le début un nombre illimité d'instances que nous pouvons associer à chaque projet, voir tableau 4.4. Nous remarquons, aussi, une différence d'à peu près de 60 secondes entre deux tests (before/after), cela s'explique par le fait que nous atteignons la limite, OpenStack (Nova Controller) rejette toute demande d'ajouter de nouvelles machines virtuelles.

Pour la deuxième catégorie d'essai (création de 500 instances), nous avons opté d'abord pour

Tableau 4.3 Création et suppression de 100 instances sans modification

Action	min (sec)	Avg(sec)	max (sec)	success Rate	Count
nova.boot_server	946.813	967.74	988.667	73%	100
nova.delete_server	251.385	269.565	287.745	73%	100
total	1198.198	1237.305	1276.412	73%	100

changer les paramètres du quota d'un nombre précis à un nombre illimité, dans chaque nouveau locataire créé, afin de se passer du problème de la limitation des instances associé à ce dernier. Après le lancement de 500 cas en parallèle, nous observons un taux de réussite en baisse (voir tableau 4.5 (success/before)). En raison du délai d'attente sur nova du nœud de calcul (compute nœud), une VIF est mise en place par Neutron, les machines virtuelles n'arrivent pas à

Tableau 4.4 Création et suppression de 100 instances après amélioration

Action	min (sec)	Avg(sec)	max (sec)	success Rate	Count
nova.boot_server	1146.092	1158.681	1171.27	100%	100
nova.delete_server	269.567	306.269	312.971	100%	100
total	1445.659	1464.949	1484.239	100%	100

Tableau 4.5 Création et suppression de 500 instances sans modification

Action	min (sec)	Avg(sec)	max (sec)	success Rate	Count
nova.boot_server	2897.592	2905.593	2913.594	67%	500
nova.delete_server	767.567	771.057	774.547	67%	500
total	3651.159	3676.650	3702.141	67%	500

être créées correctement, car elles n'ont pas une adresse IP et une adresse mac à cause de la densité de charge de travail. Cette lacune s'explique par le fait que le nombre de worker de Neutron est limité, ainsi que ces workers sont responsables de traiter les demandes de création des instances et d'allouer les adresses IP et MAC à ces derniers.

Par ailleurs, avec cette version d'OpenStack (IceHouse) nous avons la possibilité de personnaliser et de modifier le nombre de threads (worker threads) que Neutron peut utiliser, donc nous avons mis à jour Neutron plug-in en modifiant ce paramètre, afin d'améliorer sa performance (voir annexe).

Par défaut, Neutron est configuré pour correspondre à la même configuration que Nova, c'est-à-dire 1 processus par cœur - en utilisant notre propre configuration (en multipliant par 4 le nombre de worker threads). Nous avons mis 48 processus pour chaque composant dans le nœud contrôleur (48 neutron-server processes, 48 nova-api processes, 48 nova-conductor processes) tableau 4.6. Ceci nous a permis de surmonter le problème de délai d'attente de la création de VIF.

Tableau 4.6 Création et suppression de 500 instances après amélioration

Action	min (sec)	Avg(sec)	max (sec)	success Rate	Count
nova.boot_server	2747.092	2752.681	2758.27	100%	500
nova.delete_server	790.567	794.269	797.971	100%	500
total	3537.659	3546.949	3556.239	100%	500

Toutefois, il y a une autre solution : modifier en maximisant le nombre de workers par le fait de séparer les deux composants en mettant chacun dans un nœud indépendant. Ceci réduira le fait de partage des ressources physiques, entre les composants du contrôleur et le composant du Neutron.

On peut résumer les résultats obtenus sous forme de deux grands tableaux comme suite :

Tableau 4.7 Résumé de la création et la suppression de 100 instances

Action	min (sec)		Avg (sec)		max (sec)		CPU load		success		count
	before	after	before	after	before	after	before	after	before	after	
nova.boot_server	946.813	1146.092	967.74	1158.681	988.667	1171.27	7.59%	10.41%	73%	100.0%	100
nova.delete_server	251.385	299.567	269.565	306.269	287.745	312.971	0.84%	1.15%	73%	100.0%	100
total	1198.198	1445.659	1237.305	1464.949	1276.412	1484.239	8.43%	11.56%	73%	100.0%	100

Tableau 4.8 Résumé de la création et la suppression de 500 instances

Action	min (sec)		Avg (sec)		max (sec)		CPU load		success		count
	before	after	before	after	before	after	before	after	before	after	
nova.boot_server	2897.592	2747.092	2905.593	2752.681	2913.594	2758.27	40.44%	60.36%	67%	100.0%	500
nova.delete_server	767.567	790.567	771.057	794.269	774.547	797.971	4.35%	6.5%	67%	100.0%	500
total	3651.159	3537.659	3676.650	3546.949	3702.141	3556.239	44.79%	66.78%	67%	100.0%	500

Afin d'étudier et d'analyser l'effet de chaque variable indépendante (F_1, F_2, F_3) ainsi que leurs interactions sur le critère de performance (ie taux de réussite), à l'aide de Statgraphics, nous avons utilisé une analyse multifactorielle de variance (Anova). Le tableau 4.9 montre que le modèle représentant explique plus de 85% de la variabilité observée dans le taux de réussite (MONTGOMERY, 2001e). Par conséquent, nous comprenons que le seul facteur principal significatif est le nombre d'instances et toute autre interaction à un niveau de confiance de 95% tableau 4.9. Aussi, la figure 4.4 illustre les effets des variables indépendantes (F_1, F_2, F_3), leur interaction et leur effet quadratique sur les variables de réponse (taux de réussite).

Tableau 4.9 Analyse de la Variance pour taux de réussi

Source	***Somme des carrés***	***DDL***	***Moyenne quadratique***	**Rapport F**	**Proba.**
A : Number_of_instances	40.3004	1	40.3004	23.94	0.0002
B : Number_of_Tenants	0.350417	1	0.350417	0.21	0.6548
C : Number_of_User	0.0504167	1	0.0504167	0.03	0.8649
AB	7.15042	1	7.15042	4.25	0.0571
AC	0.350417	1	0.350417	0.21	0.6548
BC	1.00042	1	1.00042	0.59	0.4528
blocs	3.89333	2	3.89333	1.16	0.3412
Erreur totale	25.2537	15	1.68358		
Total(coor.)	78.3496	23			

Donc, nous pouvons conclure de cette expérience qu'OpenStack est stable et qu'il peut supporter la charge de travail tant qu'il a suffisamment de ressources pour appuyer une telle charge. Aussi, le seul facteur qui peut influer sur son rendement est le nombre d'instances créées.

De plus, nous avons fixé les deux paramètres (utilisateur et projet) puis nous avons augmenté graduellement le nombre de machines virtuelles à créer, afin de trouver le point de saturation.

En outre, nous ne pouvons pas augmenter le nombre d'instances parallèles au-delà de 800, parce que la performance diminue d'une manière significative et nous perdons la connexion

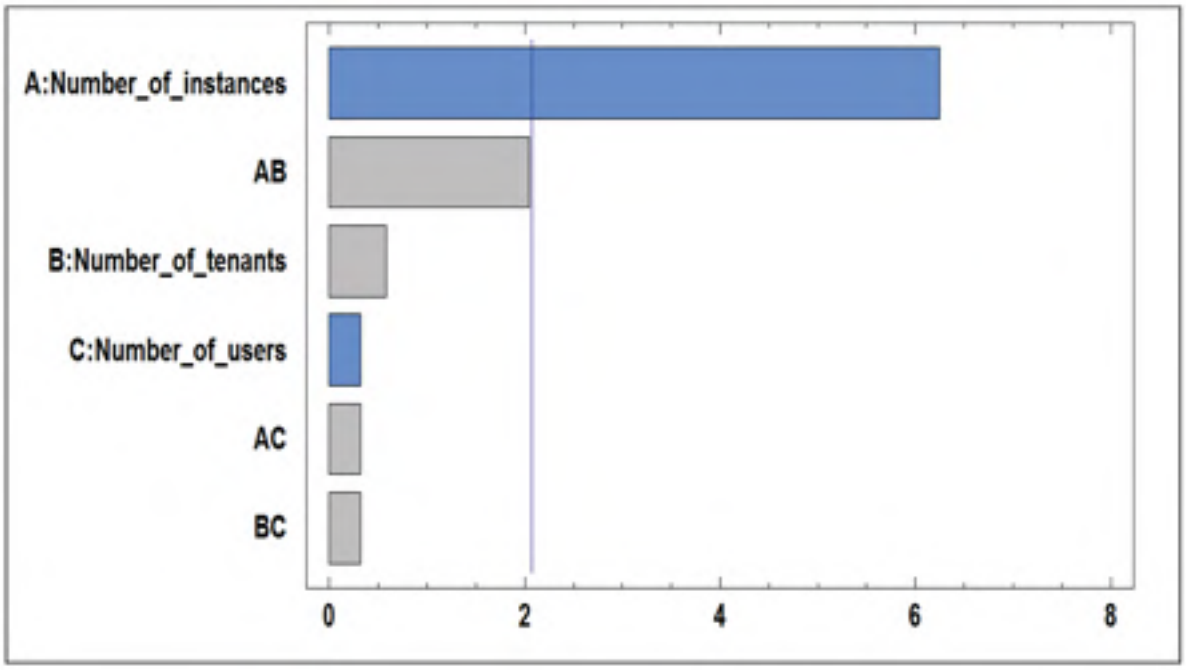

Figure 4.4 Diagramme de Pareto pour la variable dépendante du système

entre l'agent neutron et RabbitMQ. Comme par défaut, RabbitMQ gère plus de 1000 connexions simultanées, nous avons découvert que les instances ont été bloquées dans la phase du réseau en attendant pour obtenir une interface virtuelle (VIF). Normalement, lorsque la Nova envoie une requête à Neutron pour attribuer une adresse IP et Mac, Neutron lui-même demande de créer un dispositif Tap (Tap device) (voir figure 4.5). Mais, après un temps mort sans obtenir VIF, la demande est abandonnée.

Plusieurs hypothèses se sont présentées pour comprendre d'où vient le problème. Premièrement, nous avons soupçonné que la performance est en baisse à cause de la nature du scénario. Les instances sont créées, simultanées sans aucun temps d'attente pour obtenir la réponse. Ceci peut causer ce genre de dégradation, car l'envoi d'une grande charge à la fois sans attendre la réponse pourrait créer des problèmes de surcadencés (overclock).

Nous avons changé le mode du scénario de constant à en série (en créant un bloc de ma-

chines virtuelles puis nous attendons quelques secondes avant de lancer le bloc suivant et ainsi de suite) pour voir si c'est le cas ; mais la performance ne s'est pas améliorée. Par conséquent, le problème ne vient pas de la surcadencée.

Deuxièmement, nous avons pensé qu'avec la création de 800 instances et plus, nous avons déjà atteint la limite des ressources physiques. Ce qui provoque le *Overcommitting* ou *Oversubscription CPU ratio*. Dans notre cas, nous avons 24 cores par serveur avec le Hyperthreading opérationnel. Autrement dit, nous avons en total 72 cores.

– chaque hôte a deux six processeurs de base et l'hyperthreading est activé, cet hôte a vingt-quatre processeurs physiques CPU (6 cœurs x 2 processeurs x 2 threads par cœur).

Suivant les descriptions/recommandations d'OpenStack concernant l'allocation des VCPUs aux instances, avec ces caractéristiques matérielles, nous pouvons aller même à 1152 VCPU (Wiki, 2015).
– ratio d'allocation de CPU est 16 :1. Pour chaque CPU, nous pouvons lui associer 16 VCPUs.

La formule appliquée pour le nombre d'instances virtuelles sur un nœud de calcul est (OR * PC) / VC, où :

– OR est CPU Ratio de surcharge (cœurs virtuels par cœur physiques) ;
– PC est Nombre de cœurs physiques ;
– VC est Nombre de cœurs virtuels par instance.

Théoriquement, nous pouvons aller au-delà de 800 instances puisque, nous associons pour chaque virtuelle machine créée un seul virtuel CPU (VCPU).

De plus, nous trouvons dans les fiches techniques de la virtualisation comme dans (LOWE, 2015) que l'idéal est d'aller jusqu'à 75% du ratio ce qui revient à créer 864 (1152 * 75%)

instances. Même avec cette recommandation, nous n'arrivons pas à allouer plus 800 instances.

Cependant, nous remarquons que la cause provient du par-feu des règles de groupe de sécu-

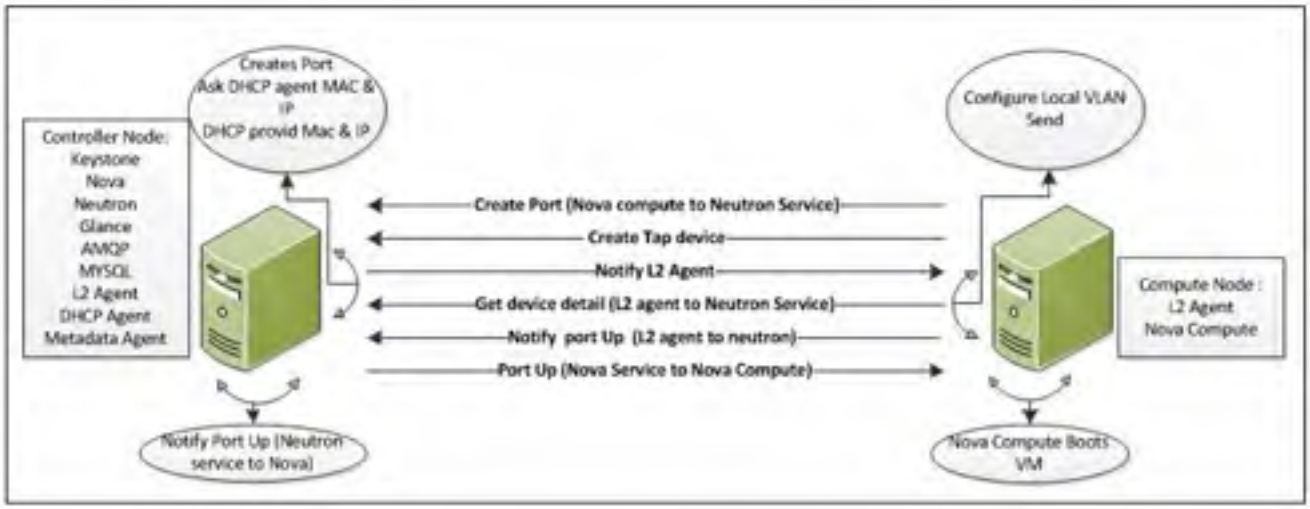

Figure 4.5 la communication entre le service de Neutron et de la Nova (21 à 23 la figure 4.3)

rité de Neutron (Neutron security group firewall) de l'agent Neutron sur le contrôleur. Après la création d'un dispositif de TAP (voir figures 4.5 et 4.6) Nova envoie RPC pour aviser l'agent DHCP, afin d'allouer des adresses IP et MAC, puis il notifie Security Group Server.

Cependant, en raison du grand nombre de demandes, agent de Neutron prend plus de temps que le temps normal requis pour vérifier et mettre à jour les règles iptables de pare-feu. D'où une dégradation croissante sur la création des instances.

En effet, l'itération avec OpenStack Compute (Nova) est plus spécifique. Lorsque Nova lance une instance virtuelle, le service communique avec OpenStack réseau (Neutron) afin de brancher chaque interface réseau virtuelle à un port particulier Figure 4.6 4.5. Comme le nombre de ports par groupe de sécurité par défaut augmente, le nombre d'entrées iptables sur le nœud Compute grandit. Pour cette raison, il y a une augmentation progressive du temps nécessaire

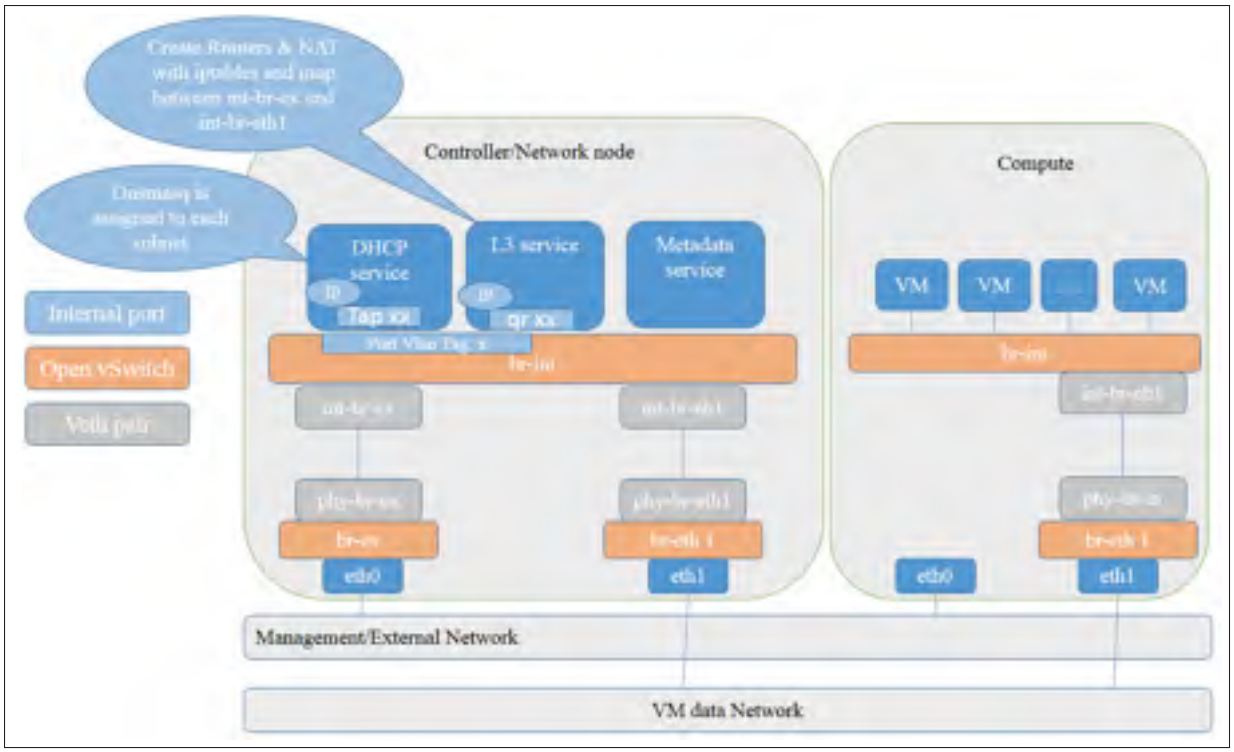

Figure 4.6 les différents éléments du neutron associer aux nœuds (controller et compute)

pour appliquer les chaines et les règles (chains and rules). Avec cette version de IceHouse, actuellement, nous utilisons une liste de compréhension pour savoir si une nouvelle chaine ou règle correspond à une déjà existante. Et cela affecte le temps de la création VIF qui explique pourquoi nous obtenons des échecs d'activation de l'instance. Cette erreur est un des points à fixer dans les versions avenir d'OpenStack. Parmi les solutions proposées, nous avons :

– changer Neutron par nova-network ;
– améliorer la méthode du gestionnaire iptables et modifier les règles ;
– analyser toute la liste dans le sens inverse pour trouver une entrée correspondante plus rapidement ;
– ajouter une nouvelle méthode qui permet de retourner l'entrée que nous recherchons ;
– augmenter le délai/temps d'attente au niveau de Neutron ;

– décomposer l'entité Contrôler et Neutron en deux entités indépendantes afin d'éliminer la possibilité de partages des ressources physiques.

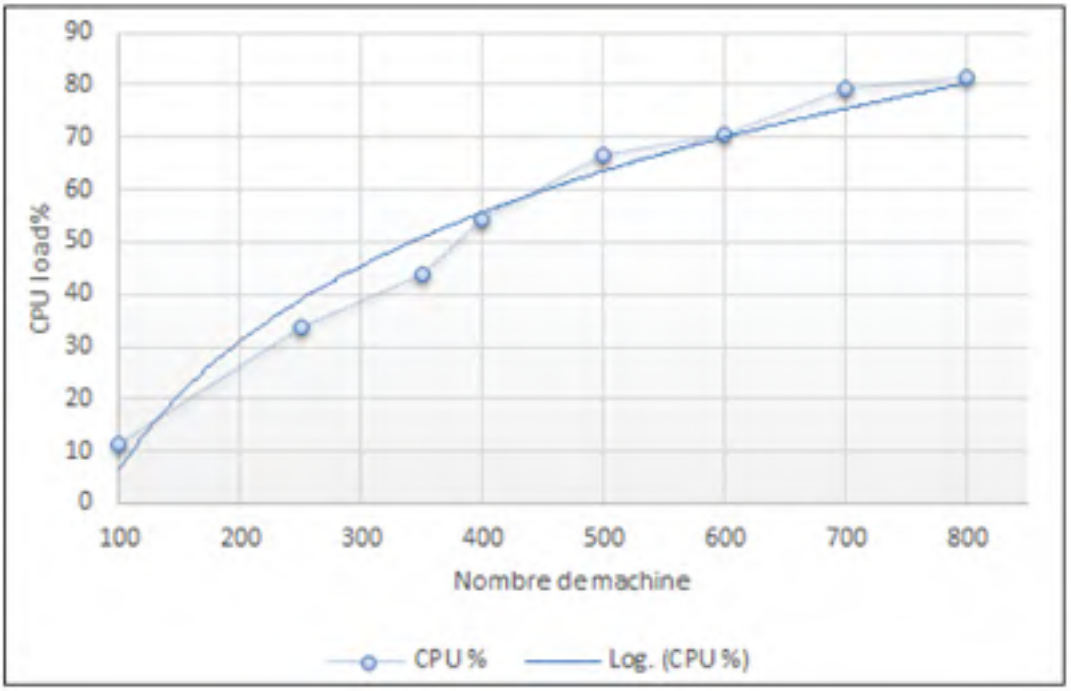

Figure 4.7 Variation du CPU basé sur la charge de travail

Afin d'évaluer le pourcentage de la charge du processeur et de l'exécution du programme Temps (PET), nous évaluons et analysons la consommation de CPU et le temps moyen nécessaire pour accomplir une tâche en fonction du nombre des instances créées et supprimées après utilisation. Pour évaluer la performance du nœud de contrôleur, tout en créant et en supprimant un ensemble d'instances et pour superviser la consommation de CPU, un nombre d'instances varie de 100 à 800. Les résultats peuvent être vus sur les figures 4.7 et 4.8 qui montrent que le nombre d'instances affecte de manière significative la charge du CPU. La charge de CPU augmente d'une manière graduelle lorsque le nombre d'instances est en dessous de 600, tandis que pour un nombre d'instances plus élevé (au-delà de 700) la charge CPU semble se stabiliser autour de 80%. En outre, il existe une corrélation entre le nombre d'instances et la charge CPU, et il atteint sa charge CPU maximum pour 800 instances. Nous expliquons le fait que le CPU a atteint sa valeur maximale de 85%, vu que nous n'avons pas pu aller au-delà de 800 instances à cause des lacunes citées au cours des discussions.

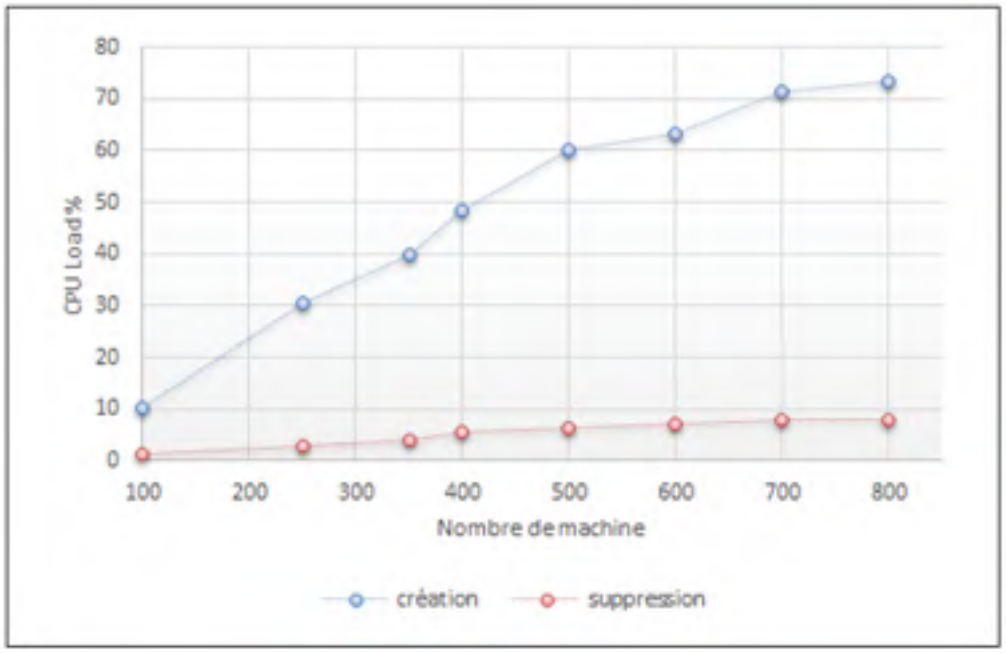

Figure 4.8 Variation du CPU basé sur la charge de travail pour la création et la suppression

Dans la deuxième partie de ces tests, le temps d'exécution du programme a été mesuré

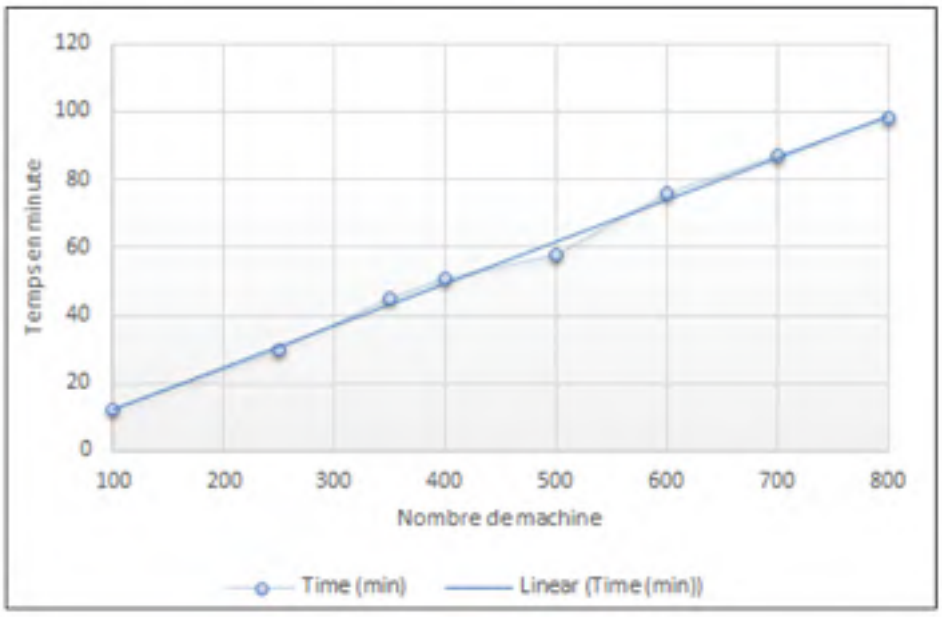

Figure 4.9 Variation du temps basé sur la charge de travail

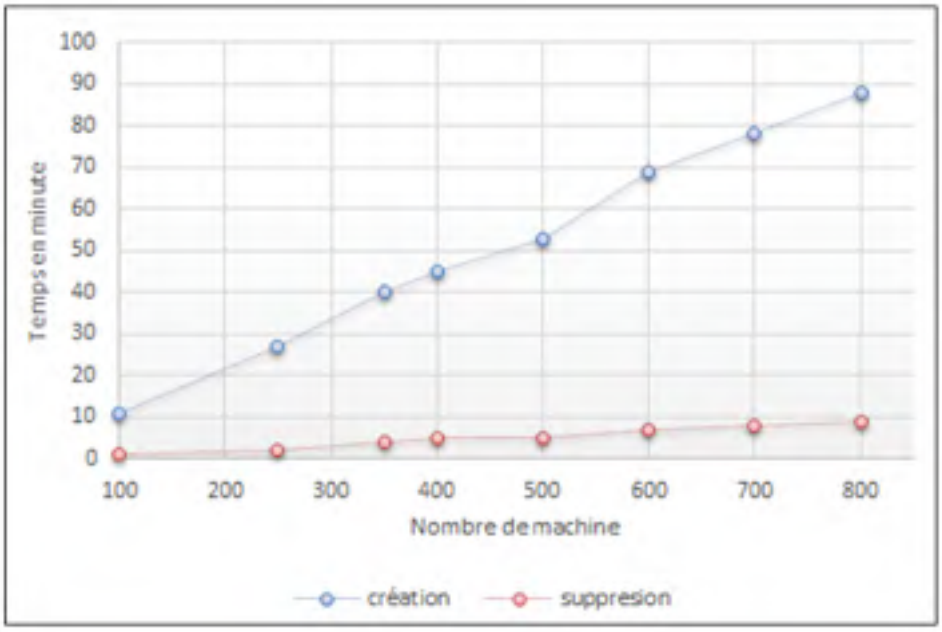

Figure 4.10 Variation du temps basé sur la charge de travail pour la création et la suppression

pendant les mêmes tests. Les résultats obtenus peuvent être vus sur la figure 4.9 et la figure 4.10 qui montre une forte corrélation entre le temps nécessaire pour réaliser le scénario et le nombre d'instances. Dans l'intervalle de 100 à 800, cette fonction augmente d'une manière linéaire.

D'après ces résultats obtenus, nous voyons clairement que le processus de provisionnement d'une machine virtuelle nécessite plus de temps et de charge de CPU que la suppression, cela s'explique que lorsqu'on crée une instance, nous faisons une sorte de collecte de données, à partir des différents composants puis nous les présentons à l'hyperviseur pour allouer cette instance. Par contre, dans le cas de la suppression nous supprimons cette instance de l'hyperviseur directement ce qui ne requière pas une grande charge de CPU et du temps pour exécuter cette tâche.

4.5 Analyse des menaces à la validité

Dans cette section, nous présenterons une discussion sur les menaces de validité. En effet, au cours de la réalisation de ce présent mémoire, nous nous sommes mis dans un contexte précis, où nous sommes limités à quelques aspects, afin de bien se focaliser dans la phase des tests et des interprétations. D'où la présence des menaces à la validité des résultats.

En effet, les critères de performance du Cloud Middleware ainsi que le banc d'essai, la configuration et le scénario pris en considération dans nos expériences, nous ont permis d'atteindre les résultats et les synthèses présentés ci-dessus.

Par ailleurs, si nous changeons ou nous rajoutons aux facteurs d'entrée (nombre d'utilisateur, nombre de projet et nombre d'instances) d'autres facteurs, comme le type et la taille d'image de l'instance, la capacité de stockage et la précision de la plage d'adresse, ceci conduit à obtenir des résultats différents de ceux présentés dans la section *Résultats et interprétation*, et qui a été explique en détail dans le chapitre *Méthodologie de Travail*.

De ce fait, nous pourrons obtenir d'autres facteurs significatifs ainsi qu'une existence des dépendances entre ces facteurs, ce qui n'était pas le cas dans notre étude. De plus, l'architecture déployée joue aussi un rôle très important. Dans notre étude, nous avons combiné le noeud du controlleur ainsi que le noeud du réseau (Neutron) dans un seul serveur. Si on sépare ces deux noeuds sur deux serveurs indépendants, nous aurons une tendances logarithmique pour la variation du CPU et le temps d'exécution peut être diminué. Mais avec certain nombre d'instances on pourrait avoir le problème de la vérification du tableau de contrôle sur le noeud Neutron (voir *Résultats et interprétation*).

Le partage des ressources physiques entre deux composants assez importants nécessite dès le départ assez de ressources physiques afin de fonctionner au maximum. Ils peuvent influencer sur la performance d'OpenStack. Nous expliquons le fait d'avoir combiné le noeud Controlleur

et le noeud réseau, à cause du manque des ressources ainsi que nous voulons avoir plus d'un noeud compute possible afin d'héberger le plus d'instances possibles. Avec les caractéristiques matériels que nous avons utilisé, on peut dire que chaque serveur peut contenir plus d'une deux centaines d'instances.

4.6 Conclusion

Un banc d'essai a été déployé pour expérimenter la méthodologie présentée dans le chapitre précédent. Par la suite, deux grandes catégories d'expérimentations ont été exécutées sur ce banc d'essai. La première a pour but d'extraire le facteur significatif parmi l'ensemble des facteurs, d'étudier leur impact sur les ressources matérielles : à savoir la charge du CPU et le temps nécessaire pour performer ces bancs d'essai.

Les résultats ont montré que les trois facteurs, à savoir le nombre des nœuds compute, le nombre de Workers et la capacité du CPU sont des facteurs déterminants pour la performance du OpenStack, alors que le nombre d'utilisateurs et le nombre des projets n'ont pas d'impact significatif sur la performance.

CHAPITRE 5

CONCLUSION GÉNÉRALE

Alors que la migration vers des solutions technologiques basées sur le Cloud afin de minimiser CAPEX/OPEX, plusieurs Cloud Middlewares ont été proposés, mais ces derniers restent en question. En effet, ces Cloud Middlewares étaient présentés afin de maximiser l'utilisation des ressources matérielles et pour plus économiser le coût de déploiement. Par conséquent, cette migration met en question le profilage de la performance du Cloud Middleware ainsi que son impact sur la couche physique.

Nous avons déterminé les métriques décrivant la performance du Cloud Middleware, soit locataire, utilisateur, instance. Puis, nous avons défini le temps d'exécution et le CPU comme métriques à superviser en phase des tests, afin de comprendre l'impact de ces derniers sur l'infrastructure. Enfin, nous avons défini une méthodologie de tests en nous basant sur les plans d' expériences afin de structurer nos tests.

Le profilage de la performance du Cloud Middleware étant difficile, ce mémoire répond à une partie de la problématique. Autant que nous sachions, ce travail présente une première initiative au profilage du Cloud Middleware. Les travaux de recherches, dans ce domaine, sont encore à leur début.

Comme discuté dans le deuxième chapitre de ce mémoire, la grande majorité des travaux tourne autour de l'analyse des applications déployées sur le Cloud , en considérant le Cloud Middleware comme acquis. Nous avons montré qu'il est possible de profiler le Cloud Middleware en se basant sur une méthodologie de travail qui s'accentue sur les plans d'expérience. Nous avons abordé cette problématique de recherche, en traitant les aspects conceptuels d'une telle architecture distribuée ainsi que la gestion des ressources dans ce type d'infrastructures.

Grâce aux expérimentations, aux résultats obtenus et aux discussions de ce mémoire de re-

cherche appliquée, nous avons identifié que la performance de OpenStack est contrôlée par deux paramètres qui sont corrélés ; le nombre de noeuds de calcul et le nombre de travailleurs (workers) à travers les serveurs Nova Controller et Neutron. Aussi, le facteur le plus significatif est le nombre de machine virtuel. Les ressources consommé par OpenStack en termes d'unité centrale de traitement CPU augmente avec le nombre d'instances créées ainsi que le temps nécessaire pour créer des instances augmente de façon linéaire avec le nombre d'instances.

Par ailleurs, nous pouvons identifier plusieurs problématiques de recherche que nous proposons comme travaux futurs. Parmi ces derniers, nous pouvons citer : agrandir le plan de travail en incluant d'autres facteurs que nous avons considérés comme stables dans nos expériences et aussi investiguer d'autre type de scénarios à savoir : la migration d'un centre de données à un autre en temps réel. Un autre onglet de recherche serait de se concentrer sur l'utilisation d'un autre type d'hyperviseur comme Xen et de faire les mêmes expérimentations afin de voir quel type de virtualisation donne de meilleurs résultats pour les Cloud Middleware.

Pour conclure ce mémoire, nous soulignons que ce dernier a fait l'objet d'une publication d'un article de conférence qui s'intitule : *Resource Consumption Assessment for Cloud Middleware* publier en EAI International Conference on Smart Sustainable City Technologies en Octobre, 2015. (voire Annexe)

ANNEXE I

INSTALLATION DE L'HYPERVISEUR KVM

KVM est un hyperviseur de type 2. Libvirt est un gestionnaire de machines virtuelles. Virt-manager est une interface graphique pour la gestion des machines virtuelles.

Installation

sudo apt-get install qemu-kvm libvirt-bin virt-manager

You need to ensure that your username is added to the group libvirtd :

sudo adduser ‘id -un‘ libvirtd

Pour ajouter votre <nom d'utilisateur> pour les groupes :

$ sudo adduser ‘id -un‘ kvm

Adding user ’<username>’ to group ’kvm’ ...

$ sudo adduser ‘id -un‘ libvirtd

Adding user ’<username>’ to group ’libvirtd’ ...

Facultatif : Installez virt-manager (interface utilisateur graphique)

Si vous travaillez sur un ordinateur de bureau, vous voudrez peut-être installer un outil graphique pour gérer les machines virtuelles.

$ sudo apt-get install virt-manager Pour plus de détail :

ANNEXE II

INSTALLATION D'OPENSTACK

L'installation d'OpenStack via Devstack ; après le téléchargement dû git :

/home/ubuntu/$git clone https ://git.openstack.org/openstack-dev/devstack

Accéder et créer un fichier nommé local.conf pour le nœud de compute.

/home/ubuntu/devstack/$ vim local.conf

[[local|localrc]]

DEST=/opt/stack

Logging

LOGFILE=$DEST/logs/stack.sh.log

VERBOSE=True

OFFLINE=False

LOG_COLOR=False

SCREEN_LOGDIR=$DEST/logs/screen

MULTI_HOST=1

Credentials

ADMIN_PASSWORD=systemets

ADMIN_PASSWORD=$ADMIN_PASSWORD

MYSQL_PASSWORD=$ADMIN_PASSWORD

RABBIT_PASSWORD=$ADMIN_PASSWORD

SERVICE_PASSWORD=$ADMIN_PASSWORD

SERVICE_TOKEN=systemlazy

HOST_IP=172.16.2.7

DATABASE_TYPE=mysql

SERVICE_HOST=@IP_ADDRESS

MYSQL_HOST=$SERVICE_HOST

RABBIT_HOST=$SERVICE_HOST

GLANCE_HOSTPORT=$SERVICE_HOST :9292

PUBLIC_INTERFACE=br4

ENABLED_SERVICES=neutron,n-cpu,rabbit,q-api,q-agt

NOVA_VNC_ENABLED=True

NOVNCPROXY_URL="http ://$SERVICE_HOST :6080/vnc_auto.html"

VNCSERVER_LISTEN=$HOST_IP

VNCSERVER_PROXYCLIENT_ADDRESS=$VNCSERVER_LISTEN

Github's Branch GLANCE_BRANCH=stable/icehouse

HORIZON_BRANCH=stable/icehouse

KEYSTONE_BRANCH=stable/icehouse

NOVA_BRANCH=stable/icehouse

NEUTRON_BRANCH=stable/icehouse

HEAT_BRANCH=stable/icehouse

CEILOMETER_BRANCH=stable/icehouse

Même chose créer un fichier local.conf pour le nœud contrôleur :

/home/ubuntu/devstack/$ vim local.conf [[local|localrc]]

DEST=/opt/stack

LOGFILE=$DEST/logs/stack.sh.log

VERBOSE=True

OFFLINE=False

LOG_COLOR=False

SCREEN_LOGDIR=$DEST/logs/screen

MULTI_HOST=1

Credentials ADMIN_PASSWORD=systemets

ADMIN_PASSWORD=$ADMIN_PASSWORD

MYSQL_PASSWORD=$ADMIN_PASSWORD

RABBIT_PASSWORD=$ADMIN_PASSWORD

SERVICE_PASSWORD=$ADMIN_PASSWORD

SERVICE_TOKEN=systemlazy

HOST_IP=@IP_ADDRESS

LIBVIRT_TYPE=kvm

Neutron options Q_USE_SECGROUP=True

FLOATING_RANGE="172.16.8.0/24"

FIXED_RANGE="10.0.0.0/24"

PUBLIC_NETWORK_GATEWAY="172.16.8.1"

Q_L3_ENABLED=True

PUBLIC_INTERFACE=eth0

Q_USE_PROVIDERNET_FOR_PUBLIC=True

OVS_PHYSICAL_BRIDGE=br-ex

PUBLIC_BRIDGE=br-ex

OVS_BRIDGE_MAPPINGS=public :br-ex

ENABLED_SERVICES=rabbit,mysql,key

ENABLED_SERVICES+=,n-api,n-crt,n-obj,n-cond,n-sch,n-novnc,n-cauth

ENABLED_SERVICES+=,neutron,q-svc,q-agt,q-dhcp,q-l3,q-meta

ENABLED_SERVICES+=,g-api,g-reg

Github’s Branch GLANCE_BRANCH=stable/icehouse

HORIZON_BRANCH=stable/icehouse

KEYSTONE_BRANCH=stable/icehouse

NOVA_BRANCH=stable/icehouse

NEUTRON_BRANCH=stable/icehouse

HEAT_BRANCH=stable/icehouse

CEILOMETER_BRANCH=stable/icehouse

ANNEXE III

AMÉLIORATION DE PERFORMANCE

Actuellement, il n'y a qu'une pid qui est exécuté pour quantum-server. Il ne suffit pas quand nous avons beaucoup d'accès à l'API.

Donc, plusieurs travailleurs pour quantum-server sont nécessaires.

Pour utiliser plusieurs processus faut modifier le nombre api_worker, en changeant le flag dans le fichier de configuration - permettra de partager le travail entre plusieurs cœurs d'une machine.

/home/ubuntu/etc./neutron$ vin neutron.conf

enlever le "#" et spécifier le nombre CPU sur api_workers au lieu de 0

========== WSGI parameters related to the API server ==========

Number of separate worker processes to spawn. The default, 0, runs the

worker thread in the current process. Greater than 0 launches that number of

child processes as workers. The parent process manages them.

api_workers = <number of CPUs> # Number of separate worker processes for service

On fera la même chose avec nova-conductor et api_compute dans le fichier de configuration de nova ajoutez :

/home/ubuntu/etc./nova$ vim nova.conf [DEFAULT]

osapi_compute_workers= <number of CPUs> # Number of workers for OpenStack API service. The default will be the number of CPUs available. [conductor]

workers= <number of CPUs> # Number of workers for OpenStack API service. The default will be the number of CPUs available.

ANNEXE IV

INSTALLATION DE COLLECTD

Actualiser l'index de package local et ensuite installer en tapant : sudo apt-get update sudo apt-get install collectd collectd-utils

Collectd

Commencez par ouvrir le fichier de configuration de collecte dans votre éditeur avec les privilèges root : sudo nano /etc/collectd/collectd.conf Hostname "hostname_of_your_machine" Pour ce guide, nous allons veiller à ce que les plug-ins suivants soient activés. Vous pouvez commenter d'autres plug-ins, ou vous pouvez travailler sur les configurer correctement si vous voulez les essayer sur votre hôte : LoadPlugin apache
LoadPlugin CPU
LoadPlugin df
LoadPlugin load
LoadPlugin memory
LoadPlugin processes
LoadPlugin write_graphite
Activer le plug-in Apache parce que nous avons installé Apache pour servir Graphite. Nous pouvons configurer le plug-in Apache avec une section simple qui ressemble à ceci : <Plugin apache>
<Instance "Graphite">
URL "http ://Hostname_or_IP/server-status ?auto"
Server "apache"
</Instance>
</Plugin>
Arriver à le plug-in Graphite. Ceci va dire au collectd comment se connecter à notre instance de Graphite. Faire la section ressembler à quelque chose comme ceci : <Plugin write_graphite>

<Node "graphing">
Host "localhost"
Port "2003"
Protocol "tcp"
LogSendErrors true
Prefix "collectd."
StoreRates true
AlwaysAppendDS false
EscapeCharacter "_"
</Node>
</Plugin>

Apache pour rapporter les Statistiques

sudo nano /etc/apache2/sites-available/apache2-graphite.conf
Au-dessous du "content", ajouter un autre bloc de sorte que Apache servira les statistiques à la page / server-status. Ajoutez la section suivante : Alias /content/ /usr/share/graphiteWeb/static/
<Location "/content/">
SetHandler None
</Location>
<Location "/server-status">
SetHandler server-status
Require all granted
</Location>
ErrorLog $APACHE_LOG_DIR/graphite-web_error.log
Recharger Apache pour avoir accès aux nouvelles statistiques : sudo service apache2 reload
http ://hostname_or_IP/server-status

ANNEXE V

STATGRAPHIC

Statgraphics est un progiciel statistique qui l'effectue et explique les fonctions statistiques de base et avancées. Le logiciel a été créé en 1980 par le Dr Neil Polhemus tout en travaillant comme professeur de statistiques à l'Université de Princeton. La version actuelle du programme, Statgraphics Centurion XVI, a été libérée à l'automne 2014. Version XVII, disponible dans les deux éditions 32 bits et 64 bits, est disponible en cinq langues : anglais, français, espagnol, allemand et italien.

Statgraphics a été le premier logiciel de statistique adapté pour le PC, le premier à introduire l'intégration des graphiques dans chaque procédure statistique, et l'auteur d'outils point par point d'assistance et d'innombrables autres caractéristiques innovantes pour simplifier les tâches.

ANNEXE VI

PROJET RALLY

Rally est un outil d'analyse comparative qui répond à la question : "Comment fonctionne OpenStack à l'échelle ?". Pour rendre cela possible, Rally unifié les multi-nœud, la vérification de nuage, l'étalonnage et le profilage d'une manière automatisé et enfichable (pluggable). Permettant de vérifier si OpenStack va bien fonctionner sous une forte charge. Ainsi, il peut être utilisé comme un outil de base pour un système de CD/CI (Continuous Deployment)/(Continuous Integration) OpenStack qui permet d'assurer la continuité d'amélioration du SLA (Service-Level Agreement), la performance et la stabilité. (Voir annexe pour le déploiement). D'après la synoptique VI-1 Rally permet de réaliser :

– Moteur de déploiement, Rally ne déploie pas OpenStack mais, est juste un mécanisme enfichable qui permet d'unifier et simplifier le travail avec différents déployés comme : DevStack, carburant, Anvil sur le matériel ou machine virtuelles que nous avons.
– Vérification utilise tempête pour vérifier la fonctionnalité d'un nuage OpenStack déployé,
– Moteur Benchmark - permet de créer la charge paramétrée sur le nuage basée sur un grand référentiel de repères.

1. Installation

exécuter les commandes suivantes :
$git clone https ://git.openstack.org/stackforge/rally
$./rally/install_rally.sh

1.1 Enregistrement d'un déploiement OpenStack en Rally

/home/devstak$. opernc admin admin /home/devstak$ rally deployment create –fromenv –name=existing

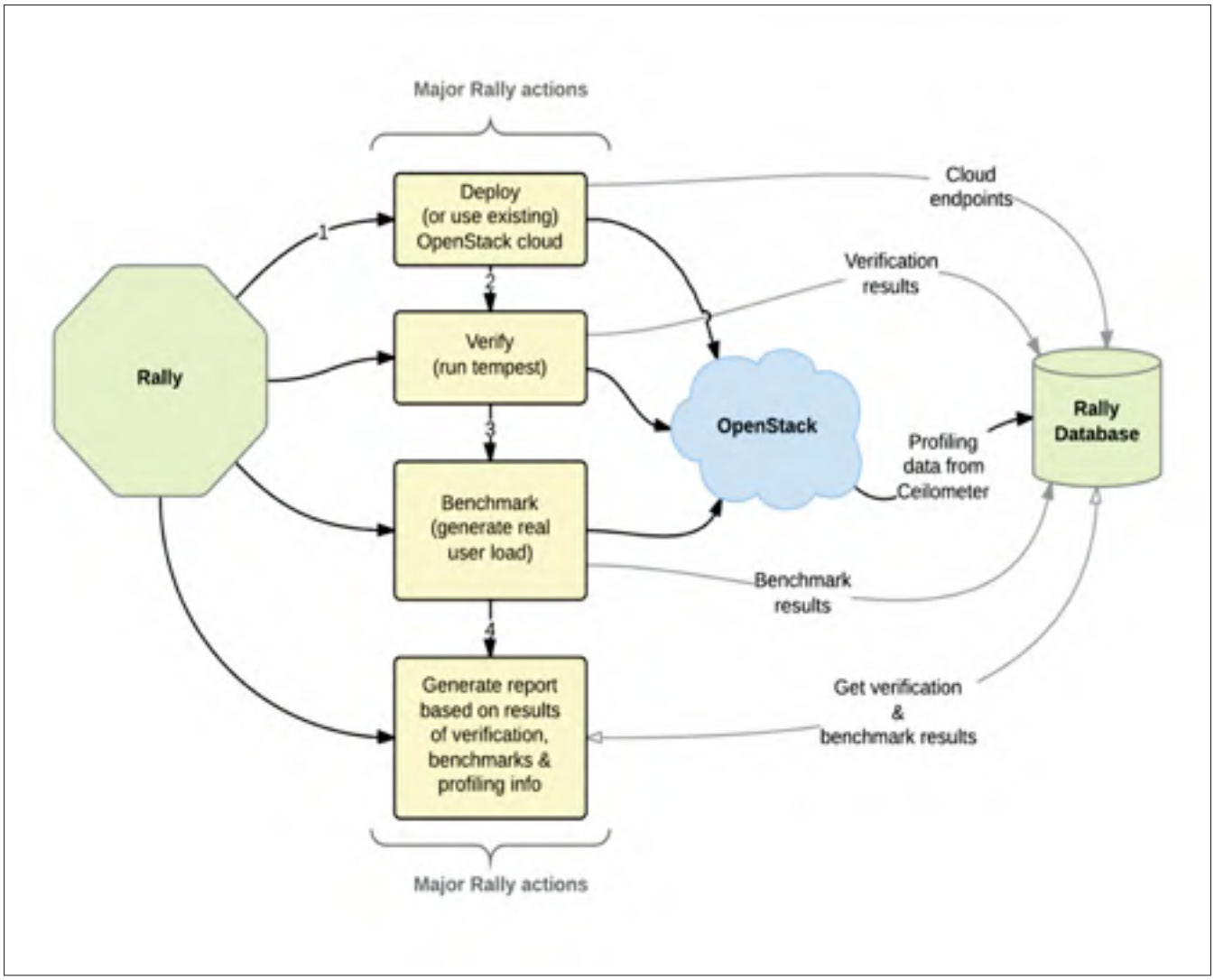

Figure-A VI-1 Quelque fonctionnalité de Rally

Maintenant que nous avons enregistré le déploiement rally avec Devstack, nous pouvons commencer le benchmarking. La séquence de repères pour être lancé par Rally doit être spécifiée dans un fichier de référence de configuration de la tâche (soit en JSON ou en format YAML). Nous avons utilisé une des tâches de référence de l'échantillon disponible en samples/tasks/scenarios, par exemple, dans notre cas été boot-and-delete.json) : { "NovaServers.boot_and_delete_server" : [

{

"args" : {

```
"flavor" : {
"name" : "m1.nano"
},
"image" : {
"name" : "cirros"
},
"force_delete" : false
},
"runner" : {
"type" : "constant/serial",
"times" : 500,
"concurrency" :500
}, "context" : {
"users" : {
"tenants" : 80,
"users_per_tenant" : 10
},
"quotas" : {
"nova" : {
"instances" : -1,
"cores" : -1,
}
"neutron" : {
"subnet" : -1,
"port" : -1,
"network" : -1,
"router" : -1
}
}
```

}

]

}

pour démarrer une tâche de référence, exécutez la commande de début de la tâche (vous pouvez également ajouter l'option -v pour imprimer plus d'informations de journalisation) : /home/ubuntu/rally/samples/tasks/scenarios/nova/$ rally task start /boot-and-delete.json

Pour encore plus de détail et de scénario voir (Ral).

ANNEXE VII

OVERCLOCKING/ SUR-CADENCEMENT

Overclocking se réfère à la configuration d'un composant matériel informatique pour fonctionner à un rythme plus rapide que ce qui était certifié par le fabricant d'origine, généralement exprimée en fréquence d'horloge donnée en mégahertz (MHz) ou gigahertz (GHz). Habituellement, la tension de fonctionnement de l'appareil overclocké est également augmentée, ce qui peut aider à maintenir la stabilité opérationnelle du composant à des vitesses accélérées (marge).

Toutefois, un dispositif semi-conducteur donné soi va générer plus de chaleur lorsqu'il fonctionne à des fréquences et des tensions plus élevées - donc la plupart des tentatives d'overclocking permettront d'accroître la consommation d'électricité et de chaleur ainsi. Une attention particulière doit être faite pour éliminer la charge de chaleur supplémentaire et veiller à ce que les composantes de soutien de livraison de puissance peut gérer les demandes d'énergie accrues demandées par le dispositif, sinon il risque de ne pas fonctionner de manière fiable ou même échouer purement et simplement à la vitesse accrue.

Par exemple, de nombreuses cartes mères avec processeurs AMD Athlon 64 limiter le taux de quatre unités de RAM à 333 MHz d'horloge. Toutefois, les performances de la mémoire est calculée en divisant la vitesse d'horloge du processeur (ce qui est un indice de base fois un multiplicateur de l'unité centrale, par exemple 1,8 GHz est très probablement 9 * 200 MHz) par un nombre entier fixe de telle sorte que, à un débit de stock horloge, le RAM irait à une fréquence d'horloge près de 333 MHz. éléments de la façon dont le taux d'horloge du processeur est fixé (généralement vers le multiplicateur), il est souvent possible d'overclocker le processeur d'un petit montant, environ 100-200 MHz (moins de 10%), et d'obtenir une fréquence d'horloge de RAM de 400 MHz manipuler (20% d'augmentation de la vitesse de la RAM, mais pas dans les performances globales du système)(the free encyclopedia, 2015).

ANNEXE VIII

RESOURCE CONSUMPTION ASSESSMENT FOR CLOUD MIDDLEWARE

Hicham Abdelfattah[1], Kim Khoa Nguyen[2], Mohamed Cheriet[3]

[1 2] Département de Génie Électrique, [3] Département de Génie Production, École de Technologie Supérieure, Laboratoire Synchromedia, 1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3 Article soumis et accepter en « EAI International Conference on Smart Sustainable City Technologies » Octobre 2015 .

Abstract

In order to provide users with relevant information about cloud operation and to be aware of cloud performance, it is essential to understand the cloud system behaviour and analyse the performance of cloud components to pinpoint on its strength and limitations. In this paper, we present a profiling approach to address this need. We profile OpenStack based on various scenarios to determine the behavior of OpenStack components and to evaluate its performance.Our experiments showed that by changing OpenStack configurations, it can support a larger amount of workload up to 800 instances. We conclude that the performance of OpenStack is controlled by two metrics which are correlated ; the number of compute nodes and the number of running instances, and the number of workers across the Nova Cloud Controller servers. Also, resource consumed by OpenStack in terms of CPU increases along with the number of instances. The current version of OpenStack cannot support more than 800 instances, and the time required for creating instances increases linearly with the number of instances.

Keywords : Virtualization, Cloud Computing, Cloud Middleware, Profiling, OpenStack, Resource Consumption.

Introduction

One of key cloud services is Infrastructure as Service (IaaS) (Peter Mel, 2011) , which is a new approach, vision for acquisition and management of physical resources. It allows the elastically and is expendable to users to add more resources based on immediate needs. This way makes IaaS suitable for companies to deploy and configure their own private Cloud. Some open-source used for private cloud are Nimbus (Nim), Eculyptus, OpenStack (Ope) and others. Facing increasing resource demand, Cloud administrator may either over-provision his cloud or reject a larger proportion of requests (which is no longer on-demand).

To better afford this kind of user requests a profiling phase is needed. Usually a profiling process is associated to a specific kind of task, we could not take the same profiling process to different problems. But mostly in term of software engineering, the purpose for profiling is to seek which part of code or program uses most resources. Very often CPU time, Memory, I/O, execution time are main profiling metrics. Furthermore, using this type of solution helps the administrator support his stack by learning more about his code. It provides an overview of each use case, gives an idea what are strength, leaking and bottleneck parts in order to intervene when it is required.

To overcome this issue, we need to build a deep measurement and profling framework for private cloud to determine the limitations and strengths before using it in a production chain. Overall, the present paper aims to characterize and profile OpenStack as Cloud Middleware. Our choice was based on prior research (Von Laszewski *et al.*, 2012) (Marshall *et al.*, 2011) (Wen *et al.*, 2012) (Caron *et al.*, 2013) (Ueda et Nakatani, 2010) (Sempolinski et Thain, 2010) and comparison between different platforms (Ren *et al.*, 2010) (Yu *et al.*, 2011) (Kutare *et al.*, 2010), and OpenStack was a recommendation to use as Cloud Middleware because of its popularity. Assuming we own a set of physical resources dedicated to allocate a set of (instances) running the same application but in different tenants, each user has his own needs to use this application.

In this paper we investigate the relationship between resources usage and Cloud Middleware performance (OpenStack as an example), which is run in a dedicated server (we do not consider cloud middleware components running on the compute nodes). Also, to determine the maximal number of instances that could supported by the stack, and study the impact of those instances on resource usage. Then, it would be beneficial to improve performance prediction to determine the success rate of user requests in order to make decision to accept or refuse a user request.
The rest of this paper is organized as follows. In section II, we outline the approach and solution. Experimental scenarios are presented with discussions in Section III. Finally, this paper is concluded in Section IV.

3. Related works

In term of performance analysis in case of sharing resources as in vitualized world (Cloud), there has been a number of efforts (Ren *et al.*, 2010) (Yu *et al.*, 2011) (Kutare *et al.*, 2010) (Do *et al.*, 2011)(O'Loughlin et Gillam, 2014) (Zabolotnyi *et al.*, 2014) for performance analysis in terms of resource consumption assessment in cloud, but so far there is no work dedicated to cloud middleware.

In (Do *et al.*, 2011), the authors discuss how to profile applications hosted by instances placed in cloud by using canonical correlation analysis (CCA) to identify the relation between application performance and system performance (resource usage). It associates canonical weight vector which represents the involvement level of system factors into correlation with application performance. Results shows that CCA is capable of finding the relationship between system and application performance. Tests show that I/O write speed is the main factor who is responsible of application performance. Also, there is a high correlation between performance of applications running on VMs and resources usage on physical nodes. They conclude that metrics with high weight in the application profile has high prediction accuracy to be key factor for performance quality.

Also in (Zabolotnyi *et al.*, 2014), authors highlighted the problem of over-provisioning of re-

source usage without using them effectively and thus resulting in financial loss and resource wastage. In order to optimize resource usage, they proposed a new profiling approach based on scheduling uniform tasks (which require CPU and RAM such as image processing applications) that are defined by developers with non-uniform resource usage and distribute them on the cloud within the defined resources limitation. The approach consists in two phases, pre-learning resource consumption by classifying tasks and then profiling them with resource requirements. After that a new task scheduled will be associated with resource usage prediction which is build up by the first step. They compare two tasks with and without profiling based scheduling. Results show that this approach could save 33% memory and 1% of time to complete the same task.

A majority of prior research in the field were focused on high level, application layer which is built on the top of architecture models by analysing and monitoring applications performance, using multiple metrics like I/O, CPU, RAM, energy consumption and others. In the application layer, prior work has taken several approaches like instrumenting code, inferring the abnormal behaviour from history logs.

4. Methodology and Test Environment

4.1 Performance metrics

In the Infrastructure as a Service layer there are two methods to lease a pool of computing resources as a set of VMs. The first method gives the ability to each cloud user to customize his VM with his own configuration or, limit this privilege to cloud administrator to set a pool of pre-configured VM. We will work with the second option, in which, all VMs are deployed and managed by the cloud administrator. This choice will help unify all VMs to one specific configuration.

Considering a set of workflow, we address two questions typically for cloud middleware performance :

1 - what the hottest processes or code region influence on middleware performance ?

2 - how does performance differ across different among of workflow ?

In general each application has some specific goals which need an amount of resources (e.g., CPU, memory, or I/O)— thus affect application performance - and resource usage. Performance metrics tend to be defined in terms of application performance. But the question of how compute performance should be defined and measured remains surprisingly contentious. This question was raised in (O'Loughlin et Gillam, 2014). In (O'Loughlin et Gillam, 2014) (O'Loughlin et Gillam, 2013) (Lilja, 2005), authors distinguish between bad and good metrics to rely on and especially those which are related to Cloud computing and HPC applications. Some of Bad metrics are Millions of Instructions Per Second (MIPS) which are calculated as the instruction count for an application divided by execution time*10^6. This rating cannot be used to compare the performance of CPU with different instruction sets. Another bad metric is Bogus MIPS (BogoMIPS) which is defined by the number of NOOP (no operation) operations a machine performs per second. On the other hand, some of good metrics like Program Execution Time (PET) are represented as clock time from the begining to the end of a program, in which the faster executing time gives higher performances scores. Also, computer Unit Processing (CPU) reflexes how much load is carried on a processor. Characteristics that define a good performance metric should follow a rule : metric has repeatability meaning it should give the same value every time we do a measurement, and it should be easy to measure and linear its value changes by the same ratio.

In our case, we have many metrics to consider like number of tenants, number of users per tenant, number of VM, VM image and VM flavor which will affect the performance of Cloud Middleware. Therefore we use an experimental design concept, especially factorial technique (explained in next subsection) in order to determine which of those metrics has the highest impact on the performance of Cloud Middleware, and then pinpoint limitations and learn how CPU and PET response vary on different load (Figure 1). For each test, we vary its work-

load and run it several times (to validate our results). The resulting performance enables us to establish links between the Cloud Middleware metrics and resource usage.

4.2 Methodological Approach

The primary objective of this experiment is to determine the success rate and pinpoint the relationship between cloud Middleware metrics. This is determined by the successful rate of scenarios with data load variation. Since we will be configuring the load per scenario, the response variable would be measured by the success rate in each scenario. For instance, a scenario is defined to create as creating a number of instances and then delete them after building them up (ready to use) and spread those instances on multiple tenants and users.
However, there are many factors that can influence in success rate. Since many of them are beyond our control, we consider only the controllable ones, such as :

– The number of instances,
– The number of tenants,
– The number of users per tenant.

To determine the relationship of these factors, and pinpoint to main factor(s), we use an experimental approach, which is a combination of simulation modeling and experimental design, including three steps, as shown in Figure VIII-1 .

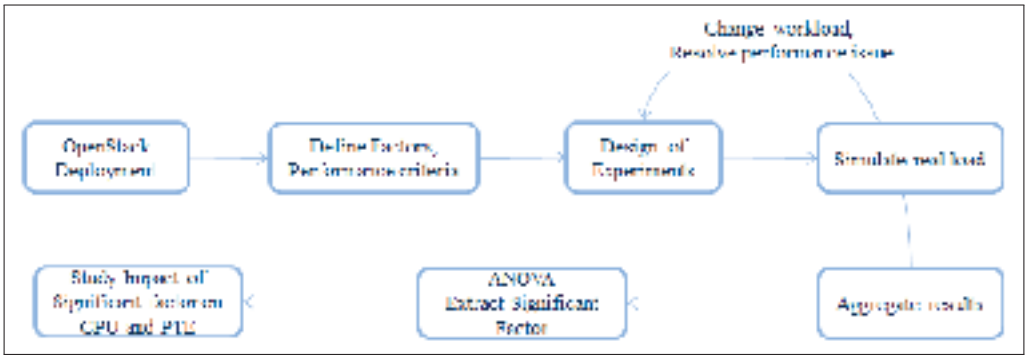

Figure-A VIII-1 the proposal methodology flowchart

1 To highlight the bottlenecks and limitation of Cloud Middleware. We simulate a real operational scenario by creating instances and then deleting them after having used. We consider the success rate as the output of the process and the input is (concurrency, tenant, users) ;
2 Data are then collected to perform a statistical analysis in order to determine the effects of the main factors, and their interactions with the dependent variable (the success rate). We aim to extract the most significant factor(s) ;
3 Determine the relationship between significant main factor(s) and/or interactions and their impact on resource usage using a regression method.

4.3 Implementation and Evaluation

In this research we focus on features which have the most important impact on OpenStack(IceHouse version (Ope)) performance. In other words, the purpose of our experimentation is to determine the limitation or bottleneck of OpenStack by creating concurrent instances within hardware capacity limitation, then find out if there is any correlation between the number of instances created, the number tenants and the number of users per tenant, and the cloud performance in terms of VM creating time. To observe how OpenStack handles concurrent requests,we set up a underlying hardware environment as shown in Figure VIII-2 :

– A physical server (node) hosting the Controller and a Compute nodes. The server has 24 core GenuineIntel 1,596 GHz, 24 GB RAM, Ubuntu 14.04.1 LTS, Kernel-based virtual machines (KVM) as hypervisor.
– Two physical servers hosting two Compute nodes, each having - 24 core GenuineIntel 1,596 GHz, 24 GB RAM, Ubuntu 14.04.1 LTS, Kernel-based virtual machines (KVM) as hypervisor.
– The nodes are connected through a 1GB Ethernet switch. The controller node is configured to do all of backend processes. Compute nodes are dedicated to host VMs.

Our scenario is to create and delete instances concurrently by creating a number of instances in parallel in order to stretch our stack to maximum and to observe how OpenStack will response. We configure our instances with a minimum requirement, using "cirros-0.3.2-x86_64-

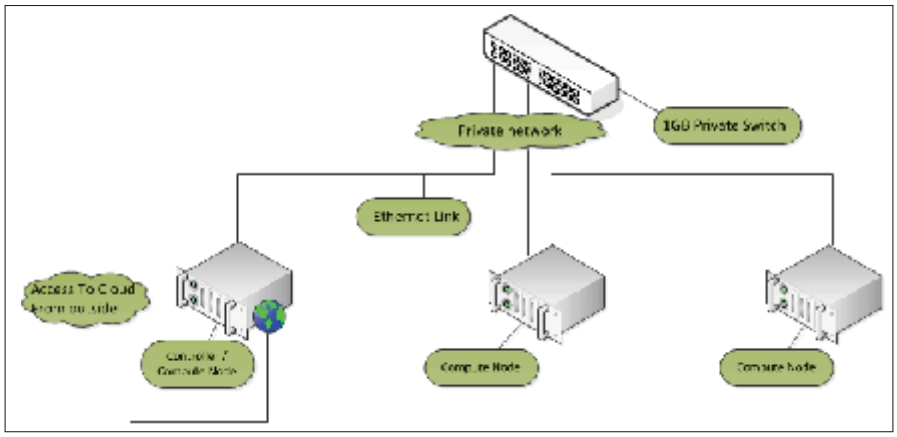

Figure-A VIII-2 Experimental cloud topology

uec" image (CirrOS cloud image) and "m1.nano" flavor (customized flavor with 1 VCPU and 64 MB of RAM). We used Rally (Ral) as tool for benchmarking.

Figure VIII-3 shows the steps taken to build up an instance. As illustrated the provisioning process comprises many stages, numbered from 1 to 23 to build up an instance. During the process the Controller node sends different requests in an order starting with verification of credentials and ending with building up instance on Compute node and going through communication inside and between components (Nova, Glance, Neutron) via RabbitMq messaging.

After launching 500 instances in parallel we observe a decreasing success rate (Table 1) due to the time-out on nova of compute node waiting for VIF to be set up. We notice that Nova daemon on the controller has specific mechanism. It seeds a number of cores equivalent to the number of sub-processes and it associates the number of instances allowed to the number of cores per each tenant. However, Neutron is configured to match what Nova does, one process per core, that is why we see timeout on compute node waiting to VIF creation to complete. In fact, in this "IceHouse version" we have the ability to customize and tweak worker threads on Neutron plugin, so we could solve this issues by scaling up more core (VCPU) to be supported per each tenant and to allow more worker threads on neutron to manage by parent process.(table 2).

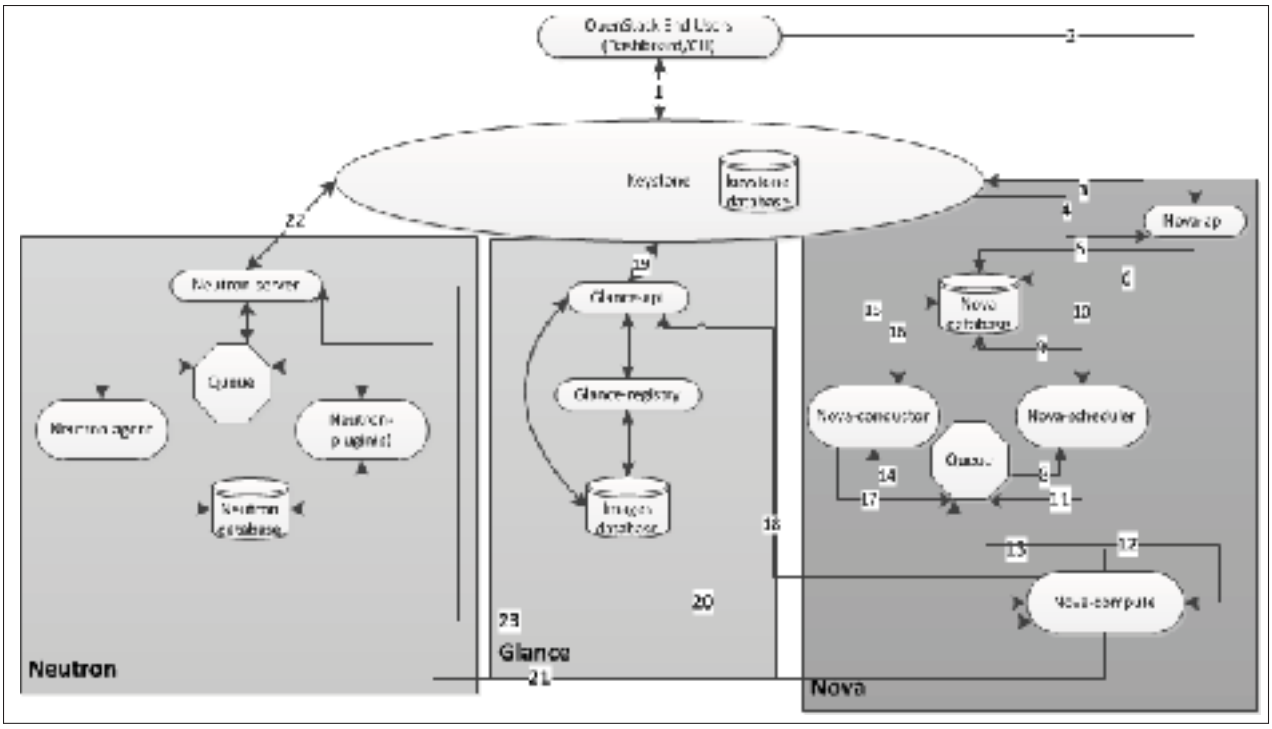

Figure-A VIII-3 Steps for instance provisioning

Furthermore, we could not increase the number of parallel instances beyond 800, because the performance drops down significantly and we lose connection between Neutron on the Controller and RabbitMQ. As RabbitMQ handles more than 1000 concurrent connections, we found out that instances were stuck at network phase waiting to get a virtual interface (VIF). Normally, when Nova sends a request to Neutron to allocates an IP and MAC address it requests to create a Tap device (see Figure 4). But after a time-out without getting VIF, the request is dropped. We suspect that performance is decreasing because concurrent instances are created

Tableau-A VIII-1 result of boot and delete scenario with 500 concurrencies

Action	min (sec)	Avg(sec)	max (sec)	success Rate	Count
nova.boot_server	32.862	161.354	331.802	75%	500
nova.delete_server	2.401	20.556	55.372	75%	500
total	35.263	181.91	347.18	75%	500

Tableau-A VIII-2 result of boot and delete scenario with 500 concurrencies

Action	min (sec)	Avg(sec)	max (sec)	success Rate	Count
nova.boot_server	24.956	161.723	256.45	100%	500
nova.delete_server	2.402	13.739	50.499	100%	500
total	27.425	175.462	269.611	100%	500

with no waiting time to get the response. Such issue could possibly comes from this kind of scenario because sending a large amount of load without waiting the response could create overlock problem. We change scenario from constant to serial mode(by creating a block of instances then we wait a couple of seconds before launching the next block) to see if it is the case ; But the performance is not improved. Therefore, the problem is not coming from the overlock.

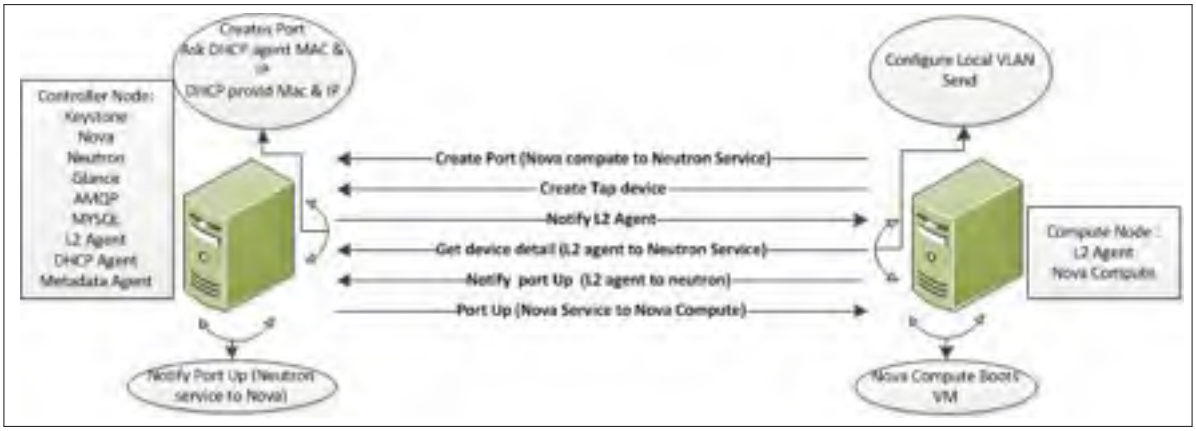

Figure-A VIII-4 communication between Neutron Service and Nova (21 to 23 of VIII-3)

However, we notice the cause is coming from Neutron security groups firewall rules of neutron agent on Controller. In Figure VIII-4 after creating a TAP device Nova sends rpc to notify the DHCP agent to allocated IP and MAC address then it notify Security Group Server. However, due to the large number of requests, neutron agent took much time than the usual required time-

out to check up and update the iptables firewall rules. The integration with OpenStack Compute (Nova) is more specific. When Nova launches a virtual instance, the service communicates with OpenStack Networking to plug each virtual network interface into a particular port. As the number of ports per default security group increases, the number of iptables entries on the Compute Node grows. Because of this, there is a gradual increase in the time taken to apply chains and rules. And it affects the VIF creation time which explains why we get instance activation failures.

After setting up our environment, we run the simulation multiple times to determine the upper bound limitation of instances created with variable concurrency and supported by Openstack with modifications mentioned above. The second phase is to deploy a suitable experimental design with a minimal set of simulation to find out if there is correlation between factors.

4.4 Correlation Factors

In this second phase, we use screening design plan to evaluate and determine the importance of each independent variable (Montgomery, 2008). We are concerned with identifying which factor(s) has important effect on the dependent variable(rate %) on the Cloud Middleware. We adopt the full factorial plane 2^3 (Montgomery, 2008) repeatedly three times. This type of full factorial design is often used and recommended as assignment model for a number of factors less than (< 5), it gives more accurate results since every interaction is estimated separately (Montgomery, 2008).

In order to study and understand the effects of independent variables (concurrency, tenant and users) on a dependent variable (success rate). We adopt a multi-factorial analysis of variances. To present the result we used Statgraphics (Sta). Figure 2 shows that the model representing Rate explains more than 83% of the observed variability in the total expected rate (Montgomery (Montgomery, 2008)). In addition, only the concurrency factor is significant at 95% level

of significance. Which means the performance of OpenStack is related to only the number of instances (Figure VIII-5). So we can conclude from this experiment that OpenStack is stable within predefined boundaries and it can carry as much workload as long as it has enough resources to support such load. Also, the only factor that can influence on its performance is the number of instances created.

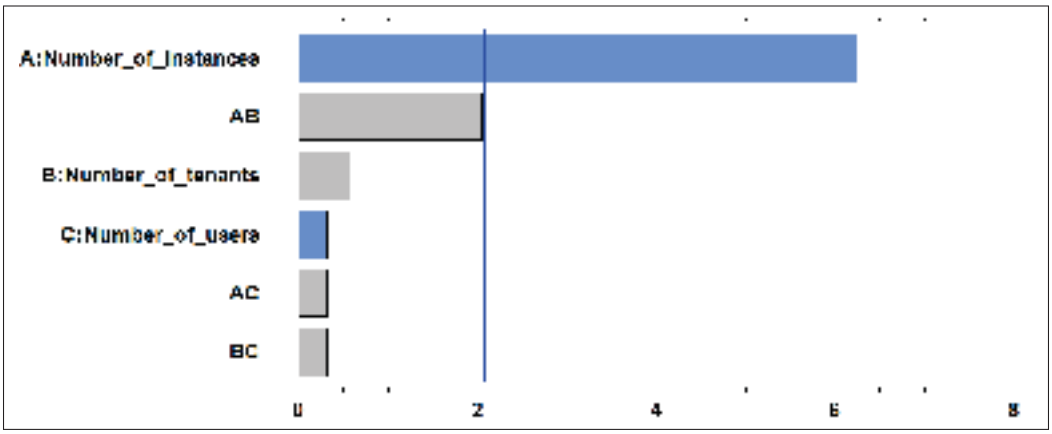

Figure-A VIII-5 Standardized effect of Pareto Chart for Rate

After determining the meaningful and influenceable metrics on the performance of cloud Middleware. We investigate the relationship between Cloud Middleware metrics and resource usage.

We consider a second testbed , as described below, to evaluate the percentage of CPU load and Program Execution Time(PET). We assess and analyze CPU consumption and the average time needed to accomplish a task as a function of the number of instances created and deleted after being used.

To assess the performance of the Controller node while creating and deleting a set of instances and CPU consumption, with a range of instances varies from 100 to 800. The purpose of this range is to maximize the CPU load. Results can be seen in Figure VIII-6 showing that the number of instances significantly affects the CPU load. The CPU load increases steadily when

the number of instances is below 600, while for a higher number of instances (beyond 700 instances) CPU load seems to stabilize around 80 %. Also,there is a correlation between the number of instance and CPU load, and it reaches the maximum CPU load for 800 instances.

In the second part of these tests, the Program Execution Time was measured while running

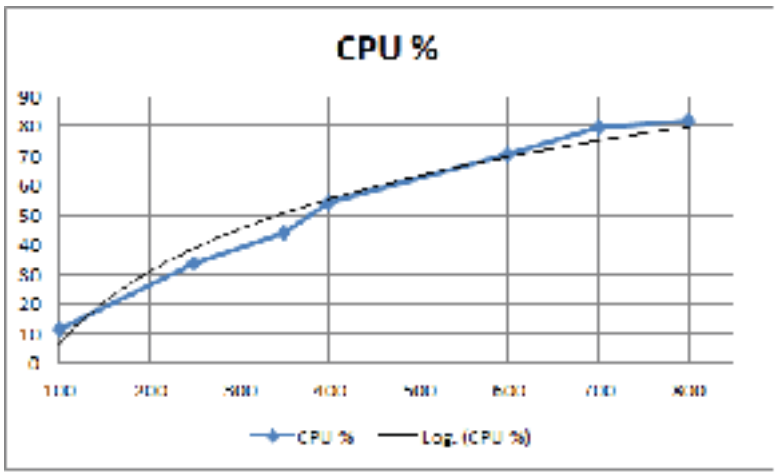

Figure-A VIII-6 Variation of CPU based on Workload

the same tests. The obtained results can be seen in Figure 7 which shows a strong correlation between the time needed to accomplish the scenario and the number of instances. Within the range 100-800 this function increases linearly.

In the third phase, we use variance analysis to validate the results. We elaborate the variation of the number of instances depending on the CPU load. Also, we use the same approach to PTE Program Time Execution to answer the question if the number of VMs has a significant impact on the CPU load and PTE.

According to the variance analysis, the adjusted R-square value indicates that the model explains over 97%, 98% of variance in CPU load (table 3), PTE(table 4) respectively. Thus, there is a relationship between the factor and the outputs. Results of the tests presented in Tables(3, and 4) show there is statistically significant interaction at a confidence level of 95% (probability values P-value of less than 0.05). A multiple linear regression model is used to describe the

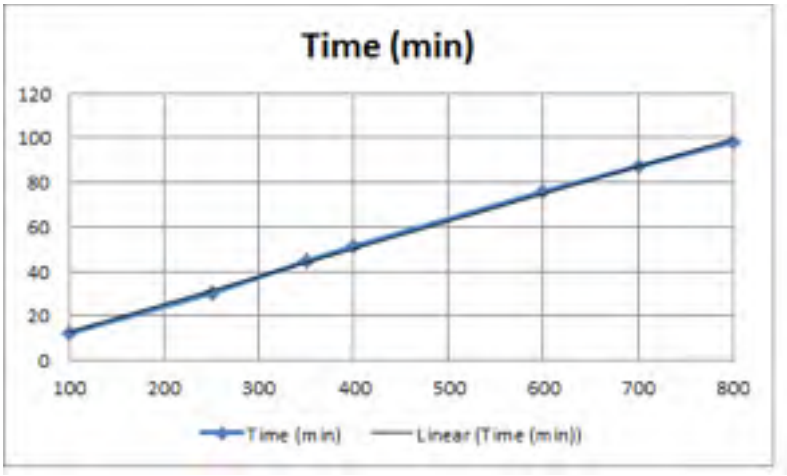

Figure-A VIII-7 Variation of time based on Workload

relation between the CPU load , PTE and the variable(VMs).

$$CPU(\%) = -2.3569 + 0.136632 * Number\ of\ VMs - 0.0000310606 * Number\ of\ VMs^2$$

$$PTE(minu) = -2.26877 + 0.139094 * Number\ of\ VMs - 0.0000173723 * Number\ of\ VMs^2$$

Tableau-A VIII-3 Analysis of Variance Order of polynomial 2

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
Model	1.12997E6	2	564984.	617.81	0.0000
Residual	16461.0	18	914.497		
Total(Corr.)	1.14643E6	20			

Tableau-A VIII-4 Analysis of Variance Order of polynomial 2

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
Model	17395.8	2	8697.89	1665.13	0.0000
Residual	94.024	18	5.22356		
Total(Corr.)	17489.8	20			

5. Conclusion

This paper highlighted two interesting results. First, we observed that it is important to be aware that we can boost OpenStack by customizing its configuration and we pointed out that OpenStack performance is related also to the number of instances created and the number of compute nodes. Secondly, through experiments we proved that the number of instances is a significant factor and there is a correlation between the number of instances created, CPU and PTE that can influence on OpenStack performance.

In future work, We plan to use XEN and KVM virtualizations in parallel and compare their results obtained with XEN results especially for PTE and CPU load consumption.

LISTE DE RÉFÉRENCES

« Eucalyptus ». http ://www.eucalyptus.com/. Accessed : 2014-03-11.

« Nimbus ». http ://www.nimbusproject.org/. Accessed : 2014-03-11.

« OpenStack ». http ://www.OpenStack.org/. Accessed : 2014-03-11.

« Rally Project ». https ://www.rally.com/. Accessed : 2014-03-11.

« Statgraphics ». http ://www.statgraphics.com/. Accessed : 2014-03-11.

Caron, E., L. Toch, et J. Rouzaud-Cornabas. 2013. « Comparison on OpenStack and OpenNebula performance to improve multi-Cloud architecture on cosmological simulation use case ».

Do, A. V., J. Chen, C. Wang, Y. C. Lee, A. Y. Zomaya, et B. B. Zhou. 2011. « Profiling applications for virtual machine placement in clouds ». In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. p. 660–667. IEEE.

Kutare, M., G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, et M. Wolf. 2010. « Monalytics : online monitoring and analytics for managing large scale data centers ». In *Proceedings of the 7th international conference on Autonomic computing*. p. 141–150. ACM.

Lilja, D. J., 2005. *Measuring computer performance : a practitioner's guide*.

Marshall, P., K. Keahey, et T. Freeman. 2011. « Improving utilization of infrastructure clouds ». In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. p. 205–214. IEEE.

Montgomery, D. C., 2008. *Design and analysis of experiments*.

O'Loughlin, J. et L. Gillam. 2013. « Towards Performance Prediction for Public Infrastructure Clouds : An EC2 Case Study ». In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*. p. 475–480. IEEE.

O'Loughlin, J. et L. Gillam. 2014. « Good performance metrics for cloud service brokers ». In *The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization*. p. 64–69. Citeseer.

Peter Mel, l. T. G. 2011. « Recommendations of the National Institute of Standards and Technology ».

Ren, G., E. Tune, T. Moseley, Y. Shi, S. Rus, et R. Hundt. 2010. « Google-wide profiling : A continuous profiling infrastructure for data centers ». *IEEE micro*, , p. 65–79.

Sempolinski, P. et D. Thain. 2010. « A comparison and critique of eucalyptus, opennebula and nimbus ». In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. p. 417–426. Ieee.

Ueda, Y. et T. Nakatani. 2010. « Performance variations of two open-source cloud platforms ». In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*. p. 1–10. IEEE.

Von Laszewski, G., J. Diaz, F. Wang, et G. C. Fox. 2012. « Comparison of multiple cloud frameworks ». In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. p. 734–741. IEEE.

Wen, X., G. Gu, Q. Li, Y. Gao, et X. Zhang. 2012. « Comparison of open-source cloud management platforms : OpenStack and OpenNebula ». In *Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on*. p. 2457–2461. IEEE.

Yu, M., A. G. Greenberg, D. A. Maltz, J. Rexford, L. Yuan, S. Kandula, et C. Kim. 2011. « Profiling Network Performance for Multi-tier Data Center Applications. ». In *NSDI*.

Zabolotnyi, R., P. Leitner, et S. Dustdar. 2014. « Profiling-Based Task Scheduling for Factory-Worker Applications in Infrastructure-as-a-Service Clouds ». In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. p. 119–126. IEEE.

BIBLIOGRAPHIE

Aceto, G., A. Botta, W. De Donato, et A. Pescapè. 2013. « Survey Cloud Monitoring : A Survey ». *Comput. Netw.*, vol. 57, n° 9, p. 2093–2115.

Amazon Web Services, I. 2015. « EC2 Amazon EC2 – Hébergement de serveur virtuel ».

Callegati, F., W. Cerroni, C. Contoli, et G. Santandrea. Oct 2014a. « Performance of Network Virtualization in cloud computing infrastructures : The OpenStack case ». In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*. p. 132-137.

Callegati, F., W. Cerroni, C. Contoli, et G. Santandrea. Dec 2014b. « Performance of multi-tenant virtual networks in OpenStack-based cloud infrastructures ». In *Globecom Workshops (GC Wkshps), 2014*. p. 81-85.

Caron, E., L. Toch, et J. Rouzaud-Cornabas. 2013. « Comparison on OpenStack and OpenNebula performance to improve multi-Cloud architecture on cosmological simulation use case ».

Corradi, A., M. Fanelli, et L. Foschini. 2014. « {VM} consolidation : A real case based on OpenStack Cloud ». *Future Generation Computer Systems*, vol. 32, p. 118 - 127.

Ericsson. 2015. « Cloud System Data Center Hardware and Equipment Manager ».

Hammadi, A. et L. Mhamdi. 2014. « Review : A Survey on Architectures and Energy Efficiency in Data Center Networks ». *Comput. Commun.*, vol. 40, p. 1–21.

Kurup, L. D., C. Chandawalla, Z. Parekh, et K. Sampat. « Comparative Study of Eucalyptus, Open Stack and Nimbus ».

Laszewski, G. V., J. Diaz, F. Wang, et G. C. Fox. 2012. « Comparison of Multiple Cloud Frameworks ». In *In Proceedings of the IEEE CLOUD 2012, 5th International Conference on Cloud Computing*. p. 24–29.

Litvinski, O. et A. Gherbi. 2013. « Experimental Evaluation of OpenStack Compute Scheduler ». *Procedia Computer Science*, vol. 19, p. 116 - 123.

LOWE, S. D. 2015. « Best Practices for Oversubscription of CPU, Memory and Storage in vSphere Virtual Environments ».

Mian, R., P. Martin, et J. L. Vazquez-Poletti. 2013. « Provisioning data analytic workloads in a cloud ». *Future Generation Computer Systems*, vol. 29, n° 6, p. 1452 - 1458.

MONTGOMERY, D. C. 2001a. JOHN WILEY & SONS, I., editor, *Design and Analysis of Experiments*, chapter 1, p. 1-8. United States of America.

MONTGOMERY, D. C. 2001b. JOHN WILEY & SONS, I., editor, *Design and Analysis of Experiments*, chapter 6, p. 218-220. United States of America.

MONTGOMERY, D. C. 2001c. Design and analysis of experiments. chapter 11, p. 427-432. United States of America.

MONTGOMERY, D. C. 2001d. JOHN WILEY & SONS, I., editor, *Design and Analysis of Experiments*, chapter 8, p. 303-317. United States of America.

MONTGOMERY, D. C. 2001e. JOHN WILEY & SONS, I., editor, *Design and Analysis of Experiments*. United States of America.

Olivier Baude, N. 2010. « La performance des processus : comment et pourquoi ? ».

Pavlovic, B. 2015. « Rally projet ».

Sahasrabudhe, S. et S. Sonawani. Oct 2014. « Comparing openstack and VMware ». In *Advances in Electronics, Computers and Communications (ICAECC), 2014 International Conference on*. p. 1-4.

Scharf, M., M. Stein, T. Voith, et V. Hilt. Aug 2015. « Network-Aware Instance Scheduling in OpenStack ». In *Computer Communication and Networks (ICCCN), 2015 24th International Conference on*. p. 1-6.

Sempolinski, P. et D. Thain. 2010. « A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus ». In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*. (Washington, DC, USA 2010), p. 417–426. IEEE Computer Society.

Shao, J. et Q. Wang. July 2011. « A Performance Guarantee Approach for Cloud Applications Based on Monitoring ». In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*. p. 25-30.

Shao, J., H. Wei, Q. Wang, et H. Mei. July 2010. « A Runtime Model Based Monitoring Approach for Cloud ». In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. p. 313-320.

Sitaram, D., P. Hallymysore, S. Harwalkar, S. Kathare, A. Srinivasan, A. Reddy, K. Kumar, et S. Sekhar. Sept 2014. « OpenSim : A Simulator of OpenStack Services ». In *Modelling Symposium (AMS), 2014 8th Asia*. p. 90-96.

Srinivasaiah, H. et N. Bhat. 2005. « Characterization of sub-100 nm {CMOS} process using screening experiment technique ». *Solid-State Electronics*, vol. 49, n° 3, p. 431 - 436.

the free encyclopedia, W. 2015. « Overclocking/sur-cadencement ».

Ueda, Y. et T. Nakatani. 2010. « Performance variations of two open-source cloud platforms ». In *Workload Characterization (IISWC), 2010 IEEE International Symposium on*. p. 1–10. IEEE.

Victor Ferraro-Esparza, M. G. et K. Olsson. 2002. « Ericsson Telecom Server Platform 4 ».

Viratanapanu, A., A. Hamid, Y. Kawahara, et T. Asami. Dec 2010. « On demand fine grain resource monitoring system for server consolidation ». In *Kaleidoscope : Beyond the Internet ? - Innovations for Future Networks and Services, 2010 ITU-T*. p. 1-8.

Voras, I., B. Mihaljević, M. Orlić, M. Pletikosa, T. Pavić, K. Zimmer, V. Paunović, S. Tomić, et al. 2011. « Evaluating open-source cloud computing solutions ». In *MIPRO, 2011 Proceedings of the 34th International Convention*. p. 209–214. IEEE.

Vázquez-Poletti, J., G. Barderas, I. Llorente, et P. Romero. 2012. A model for efficient onboard actualization of an instrumental cyclogram for the mars metnet mission on a public cloud infrastructure. Jónasson, K., editor, *Applied Parallel and Scientific Computing*, volume 7133 of *Lecture Notes in Computer Science*, p. 33-42. Springer Berlin Heidelberg. ISBN 978-3-642-28150-1. doi : 10.1007/978-3-642-28151-8_4. <http://dx.doi.org/10.1007/978-3-642-28151-8_4>.

Wiki, O. 2015. « OpensTack operations guide Chapter 4. Compute Nodes ».

Xu, Q. et J. Yuan. Nov 2014. « A Study on Service Performance Evaluation of Openstack ». In *Broadband and Wireless Computing, Communication and Applications (BWCCA), 2014 Ninth International Conference on*. p. 590-593.

Zabolotnyi, R., P. Leitner, et S. Dustdar. Aug 2014. « Profiling-Based Task Scheduling for Factory-Worker Applications in Infrastructure-as-a-Service Clouds ». In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. p. 119-126.