UNIVERSITÉ DE MONTRÉAL

ANALYSE DE PERFORMANCE DES PLATEFORMES INFONUAGIQUES

YVES JUNIOR BATIONO
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ANALYSE DE PERFORMANCE DES PLATEFORMES INFONUAGIQUES

présenté par : BATIONO Yves Junior
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. ANTONIOL Giuliano, Ph. D.., président
M. DAGENAIS Michel , Ph. D., membre et directeur de recherche
M. KHOMH Foutse, Ph. D., membre

DÉDICACE

I dedicate this work to my Lord and Savior,
Jesus Christ,
through whom all things are possible.

Je dédie cette œuvre à mon Seigneur et Sauveur,
Jésus-Christ,
par qui tout est possible.

REMERCIEMENTS

Je tiens d'abord à remercier mon directeur de recherche, le Professeur Michel Dagenais, pour m'avoir donné l'opportunité d'accomplir cette maîtrise. Le soutien et les conseils qu'il m'a offerts ont été essentiels pour mener à bien ce travail.

Je remercie tous mes collègues du laboratoire DORSAL pour leur esprit d'équipe et le partage des connaissances qui favorisent grandement la réalisation des projets de recherche. Je remercie particulièrement Francis pour sa disponibilité et son expertise dont il m'a fait profiter. Merci à Geneviève et à Mohamad de m'avoir guidé tout au long de mes travaux.

Merci aussi à Ericsson, à EfficiOS et au Conseil de Recherches en Sciences Naturelles et en Génie du Canada (CRSNG) pour leur soutien financier qui a permis l'accomplissement de cette étude.

Finalement, j'aimerais remercier ma famille, en particulier ma mère pour m'avoir poussé avec une grande patience dans mes études. Un grand merci à mes amis pour leur support constant ces deux dernières années.

RÉSUMÉ

L'usage des services infonuagiques connait un essor indéniable au sein des entreprises ces dernières années. Ils exposent, au travers d'Internet, un ensemble de technologies qui donnent accès à des ressources informatiques. Ces technologies permettent de virtualiser les machines physiques, et de fournir aux utilisateurs des ressources virtuelles isolées les unes des autres. Si ce mécanisme d'isolement offre une forme de garantie pour la sécurité des données, il peut cependant engendrer des anomalies de performance. En effet, les systèmes virtuels ont l'illusion d'avoir un accès exclusif à l'hôte et chacun l'utilise sans tenir compte des besoins des autres. Cela entraine des interférences et une baisse de performance des technologies de virtualisation.
Pour superviser les services d'une plateforme infonuagique, il est fréquent d'utiliser des applications de gestion. Ces applications ont la particularité de simplifier les interactions des utilisateurs avec les services de l'infrastructure. Cependant, elles peuvent être une source d'incohérence et causer des anomalies d'exécution des requêtes lorsqu'elles sont mal configurées. Nous nous intéressons, ici, aux troubles liés à l'utilisation d'OpenStack comme application de gestion.

L'objectif de cette étude est de fournir aux administrateurs, un outil pour surveiller le fonctionnement des opérations dans le nuage et pour localiser d'éventuelles altérations de performance, aussi bien au niveau des couches applications que des couches de technologies de virtualisation. Nous basons notre approche sur le traçage, qui permet de comprendre efficacement les détails d'exécution d'une application. En traçant simultanément les différentes couches de l'infrastructure, il est possible de suivre les requêtes des utilisateurs et de déterminer, de façon précise, les performances des services déployés.
Nous utilisons LTTng, un outil de traçage reconnu pour les latences relativement faibles qu'il induit aux programmes diagnostiqués, contrairement aux autres traceurs. Il sera utilisé pour tracer les espaces utilisateurs et noyaux des machines de la plateforme. Les traces des différents systèmes seront collectées et agrégées dans une machine dédiée à l'analyse des performances. L'administrateur pourra alors obtenir un bilan d'utilisation des ressources, détecter les anomalies de fonctionnement des services et par la suite prendre des mesures pour remédier aux problèmes.

ABSTRACT

Cloud computing usage has experienced a tremendous growth in companies over the past few years. It exposes, through the Internet, a set of technologies granting access to computing resources. These technologies virtualize physical machines to provide virtual resources which are isolated one from another. If this isolation mechanism is a guarantee for data security, it can cause a serious drop in performance. Indeed, virtual systems have the illusion of an exclusive access to the host's resources and they use them without considering the needs of others. This causes some interferences and decreases the performance of guest environments. Some applications, known as cloud operating systems, are commonly used to supervise cloud computing platforms. These applications simplify the interactions of the users with the infrastructure. However, they can cause faulty executions when misconfigured. Here we will focus on issues related to the use of Openstack as a cloud management application.

The objective of this study is to provide administrators with a tool to monitor cloud tasks and locate potential drops of performance in both application and virtualization layers. Our approach is based on tracing, to produce detailed information about service operations. By tracing the various layers of the infrastructure simultaneously, it is possible to follow user requests and accurately determine the performance of deployed services.
We use LTTng, a high-performance tracer, with very low impact on system behavior when tracing is enabled. It will be used to investigate all the host user space and kernel space executions. The traces will be collected and aggregated into a dedicated system to perform the analysis. The administrator can then obtain a resource utilization report, and be able to identify service troubles and subsequently take action to correct the problems.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AMQP	Advanced Message Queuing Protocol
ARP	Address Resolution Protocol
CTF	Common Trace Format
DHCP	Dynamic Host Configuration Protocol
DORSAL	Distributed Open Reliable Systems Analysis Lab
IO	Input Output
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
KVM	Kernel-based Virtual Machine
LTTng	Linux Trace Toolkit :next generation
NFV	Network Function Virtualization
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
NRPE	Nagios Remote Plugin Executor
NTP	Network Time Protocol
OvS	Open vSwitch
RCU	Read-Copy-Update
SDN	Software Defined Networking
SHT	State History Tree
SNMP	Simple Network Management Protocol
VM	Virtual Machine

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

1.1 Définitions et concepts de base

1.1.1 Introduction à l'infonuagique

Depuis son apparition, l'infonuagique a été définie de différentes façons par plusieurs organismes. Selon l'institut national des normes et de la technologie [1] des États-Unis, l'infonuagique est «un modèle d'accès via un réseau de télécommunications, à la demande et en libre-service, à des ressources informatiques partagées configurables». Il permet de délocaliser le traitement et le stockage de l'information sur des serveurs distants, contrairement au modèle traditionnel où tout le traitement se déroule sur la machine locale. Cette technologie présente de nombreux avantages économiques pour les entreprises et cela a largement contribué à sa popularisation. En effet, les utilisateurs des services déployés dans le nuage ne sont plus obligés d'investir dans une infrastructure informatique ou d'acheter des licences d'utilisation de logiciels. En outre, ils bénéficient d'une mise en œuvre et d'un déploiement rapide de leurs services, et d'une souplesse d'utilisation d'un ensemble de ressources informatiques, qu'individuellement ils pourraient ne pas se permettre d'acquérir.
Si, pour l'utilisateur, l'accès aux services sur le nuage est grandement simplifié, pour le fournisseur, la mise en œuvre et la surveillance de l'activité de la plateforme est une autre paire de manches. Aussi, avec l'usage de matériel hétérogène et la complexification des technologies de l'information, il n'est pas toujours aisé de fournir une bonne performance aux consommateurs, ce qui a donné naissance à de nombreux outils pour la mesure de la qualité et pour la détection des anomalies des infrastructures. Dans la suite de notre étude, nous nous intéressons de près à l'analyse des anomalies dans les plateformes infonuagiques.

1.1.2 Modèles de Plateformes infonuagiques

Ils existent différentes formes de plateformes infonuagiques[Rimal et al. (2010)] ayant chacune, des avantages spécifiques pour les utilisateurs.

Infrastructure-service : désigné sous l'expression *Infrastructure as a Service (IaaS)*. Ce modèle propose un ensemble de serveurs, d'équipements de stockage et de réseautiques qui sont accessibles à la demande. La gestion et le contrôle des ressources matérielles ne relèvent cependant pas de l'utilisateur. Pourtant, celui-ci a le contrôle sur la quantité de

1. National Institute of Standards and Technology (NIST) : est une agence du département du Commerce des États-Unis

ressources qu'il utilise et peut décider à tout moment de l'accroitre ou de la diminuer, selon ses besoins. Ainsi, l'utilisateur a la possibilité de réserver de l'espace disque, de créer des machines virtuelles et d'y exécuter les programmes de son choix.

Plateforme-service : ou *Platform as a Service (PaaS)*. Ce modèle de service inclut, en plus des infrastructures matérielles, un ensemble d'applications, de base de données et des outils de développement qui sont accessibles via des interfaces de programmation pourvues par le fournisseur du service. Ce modèle permet à l'utilisateur de déployer ses propres applications qu'il contrôle à volonté. Cependant, la gestion des infrastructures sous-jacentes, comme les équipements physiques ou les serveurs virtuels, ne lui est pas accordée.

Logiciel-service : dénommé *Software as a Service (SaaS)*, il fournit un ensemble de logiciels qui sont accessibles à partir d'un navigateur web. Ce modèle est utilisé pour déployer des programmes de gestion de ressources humaines, ou plus communément, des applications de messagerie ou des logiciels collaboratifs. L'utilisateur n'est plus contraint d'investir dans l'achat d'équipements informatiques ou de licences logicielles. Ce type de service est appelé à remplacer les programmes traditionnels qui sont installés localement sur la machine de l'utilisateur.

1.1.3 Machine virtuelle

Dans le domaine de l'infonuagique, une machine virtuelle est un conteneur capable d'exécuter son propre système d'exploitation comme s'il s'agissait d'un véritable ordinateur. Ce système d'exploitation, dit système invité, dispose de ses propres ressources virtuelles (CPU, carte d'accès réseau et disque dur) et s'exécute de façon isolée de l'environnement physique, tout en ayant l'illusion d'avoir un accès direct et exclusif à celui-ci. Les machines virtuelles permettent de s'abstraire des spécificités matérielles de la machine hôte et de faciliter l'exécution d'applications qui ne sont pas nativement conçues pour certaines caractéristiques physiques. Pour partager efficacement les ressources informatiques dont ils disposent, les fournisseurs de services infonuagiques mettent à la disposition de leurs clients, des machines virtuelles. L'isolement de leur système d'exploitation est censé éviter qu'un client n'entre en possession des données des autres utilisateurs, garantissant ainsi une certaine sécurité de l'information disponible sur la machine physique.

1.1.4 Introduction au traçage

Le traçage est une technique de journalisation permettant d'obtenir de l'information concernant l'exécution d'une application. Ces données vont aider les programmeurs à détecter des problèmes intervenant pendant le fonctionnement du logiciel tracé. Une trace est donc un bilan détaillé de l'exécution d'un système informatique.

La première étape du traçage consiste à identifier les fonctions qui nous intéressent dans notre programme et à y insérer des points de traces. Cette procédure est communément appelée instrumentation. Un point de trace est une portion de code qui permet de retourner l'état du programme lorsque la fonction instrumentée est exécutée. Dans ce mémoire, nous faisons référence au terme évènement pour décrire un point précis de l'exécution d'une application. Chaque évènement est associé à une date et à une action qui décrit la procédure exécutée par le programme à cette date-ci. À l'exécution du code d'un point de trace, un évènement est enregistré pour indiquer l'état du système analysé. L'ensemble des évènements collectés forme une trace du fonctionnement du programme.
Dans une machine, chaque programme peut s'exécuter à deux niveaux différents : niveau *utilisateur* et niveau *noyau*. Le système d'exploitation s'exécute en espace noyau tandis que les autres programmes sont écrits pour fonctionner en espace utilisateur. Cependant, les appels de ceux-ci à des fonctions du système d'exploitation se déroulent au niveau noyau.
Traçage noyau - Le noyau Linux est largement instrumenté, ce qui permet aux outils de traçage de pouvoir collecter des traces d'exécution au niveau système. L'analyse des appels systèmes permet d'observer le comportement des programmes.
Traçage niveau utilisateur - Lorsque les traces noyaux sont insuffisantes pour diagnostiquer les performances d'une application, le programmeur peut modifier son code source pour y insérer des points de trace. Cela lui permet de retrouver l'état du programme durant son exécution.

Dans cette étude, on désignera par *traçage noyau*, le traçage effectué en espace noyau (noyau Linux) pour collecter principalement les informations des appels systèmes. Par contre, le *traçage au niveau utilisateur* est celui réalisé sur les autres programmes qui s'exécutent en espace utilisateur.

1.2 Éléments de la problématique

Les plateformes infonuagiques contiennent un grand nombre d'équipements informatiques, souvent hétérogènes, qui interagissent pour fournir un service aux utilisateurs. Toutefois, plus la plateforme dispose de machines, plus la détection des anomalies devient complexe. Des pannes ou des soucis de performance peuvent survenir dans n'importe quel serveur ; il faudrait alors avoir des outils performants qui analysent efficacement les infrastructures, tout en limitant au maximum leur impact sur celles-ci. À cet effet, de nombreux outils de monitorage permettent de détecter les anomalies des systèmes et de prévenir l'administrateur des défaillances. Cependant, la complexité des mécanismes de virtualisation rend certaines défaillances imperceptibles pour la plupart des outils. Comme un des grands avantages de la virtualisation est l'isolement des environnements invite et hôte, il est difficile de détecter une défaillance au niveau de la machine virtuelle en se contentant de scruter la machine physique. Plusieurs travaux sur l'analyse des systèmes distribués et spécifiquement sur les machines virtuelles ont démontré que le partage des ressources sur l'hôte pouvait engendrer des interférences et causer des problèmes de performance. Chaque machine virtuelle suppose un accès exclusif au matériel et l'utilise autant qu'il en a besoin sans tenir compte des autres.

L'application de gestion du service infonuagique peut aussi être une source d'incohérence entre les machines de la plateforme et causer des erreurs d'exécution. Donc, d'une part une défaillance peut survenir sur une machine quelconque dans le réseau, et d'autre part cette panne peut se situer au niveau de l'application de gestion, de la technologie de virtualisation ou au niveau du système d'exploitation de la machine physique. Dès lors, un outil de diagnostic devrait avoir la capacité d'agréger les informations de toutes les machines du parc informatique et de détecter avec précision la source du problème.
Le traçage s'est révélé être un bon moyen pour acquérir des données utiles sur l'exécution d'un système distribué. L'union et l'analyse des traces des différentes machines ne sont pas toujours aisées, puisqu'il faut réussir à relier les requêtes effectuées en amont du service (application de management) et leur exécution en aval sur les machines physiques (système d'exploitation hôte). Notre problématique est de fournir à l'administrateur un moyen de localiser les sources de défaillances, de n'importe quelle couche d'une plateforme infonuagique.

1.3 Objectifs de recherche

Considérant les éléments évoqués dans la section [1.2], nous nous proposons de répondre à la question de recherche suivante :

est-il possible, par agrégation des traces de l'espace noyau et des traces de l'espace utilisateur, d'investiguer les sources de baisse de performance et de détecter les troubles de fonctionnement des services infonuagiques ?

Pour répondre à cette problématique, nous avons défini les objectifs de recherche suivants :

1. Proposer un mécanisme de traçage et instrumenter les services infonuagiques en espace utilisateur.
2. Établir un modèle permettant de retracer les requêtes des utilisateurs et leur exécution sur la plateforme, en reliant les traces de l'espace utilisateur et les traces du noyau.
3. Implémenter le modèle et fournir un outil utilisable.
4. Valider que l'outil obtenu permet d'analyser les états des activités du service infonuagique et de détecter les sources de baisse de performance.

1.4 Plan du mémoire

Le mémoire est structuré comme suit : le chapitre [2] propose un état de l'art sur le traçage des systèmes et des technologies de virtualisation. Dans le chapitre [3], nous présentons notre méthodologie pour répondre à la problématique de notre étude. Ensuite, dans le chapitre [4], nous exposons l'article de journal : *"Cloud Service Performance Diagnosis Using Multi-Layer Tracing"*. Puis, nous établissons une discussion des résultats obtenus dans le chapitre [5], pour enfin conclure le mémoire dans le chapitre [6], et proposer des recommandations pour améliorer la solution.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.1 Traçage et monitorage des systèmes informatiques

Dans cette section, nous abordons les concepts de base de la virtualisation des systèmes informatiques qui sont traités dans notre recherche. Pour comprendre le fonctionnement et analyser les performances des technologies de virtualisation, il est possible d'utiliser des logiciels comme les traceurs. Il existe de nombreux outils pour le traçage de programmes informatiques, mais nous ne décrivons que les caractéristiques de certains d'entre eux.

2.1.1 Outils de traçage

Ftrace

Ftrace[Rostedt (2009)] est un outil de traçage des systèmes informatiques. Il est destiné à aider les programmeurs à comprendre le fonctionnement de leurs systèmes et à localiser les causes de dégradation de performance. Il permet de tracer exclusivement les évènements comme les appels systèmes et les fonctions d'ordonnancement en espace noyau. Ftrace enregistre les données au format texte, mais il est possible de le configurer pour obtenir des traces au format binaire. L'usage de Ftrace n'est pas toujours adéquat en milieu de production, car il peut induire des latences non négligeables aux systèmes tracés à cause des verrous qu'il utilise pour se synchroniser.

LTTng

Nous utilisons LTTng[Desnoyers (2009)] pour tracer l'espace utilisateur et l'espace noyau des machines de notre plateforme. LTTng se présente comme un ensemble de modules pour le noyau Linux et de codes qui sont chargés dans les applications utilisateurs. Il permet de collecter les évènements avec un faible coût et de comprendre les interactions entre les composants du système tracé. Il utilise un mécanisme de synchronisation RCU (Read-Copy-Update) pour optimiser la collecte des évènements. Cette procédure permet d'éviter l'usage de verrous lors d'accès concurrents à des ressources partagées. En effet, le verrouillage est en général la principale cause de latences lors du traçage. Lorsqu'un programme est instrumenté, les points de trace sont seulement activés pour le traçage et désactivés après pour éviter de perturber l'exécution de celui-ci. Les traces sont produites dans le format CTF (Common Trace Format)[Desnoyers] et stockées sur disque. Depuis sa version 2, LTTng propose un mécanisme qui permet de tracer les applications et d'envoyer les données sur le réseau, vers

une autre machine dédiée à la collecte et à l'analyse des traces. Cette procédure sera largement utilisée dans notre étude pour tracer les machines.

2.1.2 Lecture et visualisation de traces

Babeltrace

Babeltrace est outil de visualisation de traces générées au format CTF. Il permet de les convertir vers un format texte lisible par un humain. D'une manière générale, il est possible de créer des modules permettant de convertir n'importe quel type de format de trace vers un autre format désiré. Babeltrace est souvent utilisé avec des scripts pour calculer les métriques des traces collectées.

Trace Compass

Trace Compass est un outil développé en java par Ericsson et le laboratoire DORSAL *(Distributed open reliable systems analysis lab)* de l'École Polytechnique de Montréal. Il supporte plusieurs formats de trace et permet d'analyser les données de l'espace noyau et de l'espace utilisateur fournies par LTTng. Trace Compass présente l'information extraite des traces, dans des interfaces graphiques plus ergonomiques que le format texte, facilitant ainsi l'investigation. De plus, il offre à l'administrateur la possibilité de créer ses propres interfaces adaptées aux comportements des applications qu'il diagnostique.

2.2 Concepts de base de la virtualisation

2.2.1 Émulateur

Un émulateur est un logiciel qui reproduit le comportement d'un système informatique. Lorsque nous désirons faire fonctionner une application sur une autre architecture que celle d'origine, il suffit d'installer un logiciel émulateur et de lancer l'application désirée. Cependant, l'émulateur ajoute un surcoût non négligeable lors de l'interprétation des instructions. Cela le rend moins performant que le système qu'il veut imiter. Un exemple populaire d'émulateur est QEMU[Bellard (2005)] ; il permet d'exécuter un système d'exploitation conçu pour des environnements différents de la machine hôte.

2.2.2 Hyperviseur

Un hyperviseur est un logiciel qui s'exécute sur la machine hôte pour contrôler le cycle de vie des machines virtuelles. Il se présente comme un gestionnaire de machines virtuelles : *Virtual machine Monitor (VMM)*. Il a pour rôle d'assister et de contrôler les accès des systèmes invités aux ressources de l'hôte. Deux types[Desai et al. (2013)] d'hyperviseurs sont généralement utilisés :

hyperviseur type 1 - il s'exécute directement sur la couche matérielle de la machine hôte ; c'est la couche qui se situe entre l'environnement invité et la couche matérielle. Parmi les hyperviseurs de type 1 on peut citer XEN[Barham et al. (2003)] et VMWare ESX[Muller and Wilson (2005)].

hyperviseur type 2 - il s'exécute au sein du système d'exploitation hôte. Dans ce cas, l'Hyperviseur est une couche qui sépare les machines virtuelles du système d'exploitation hôte. KVM[Kivity et al. (2007)] et VirtualBox[Oracle (2011)] sont des exemples d'hyperviseurs de type 2.

Nous utilisons KVM comme hyperviseur dans notre infrastructure. Il est aujourd'hui intégré au noyau Linux sous forme de module.

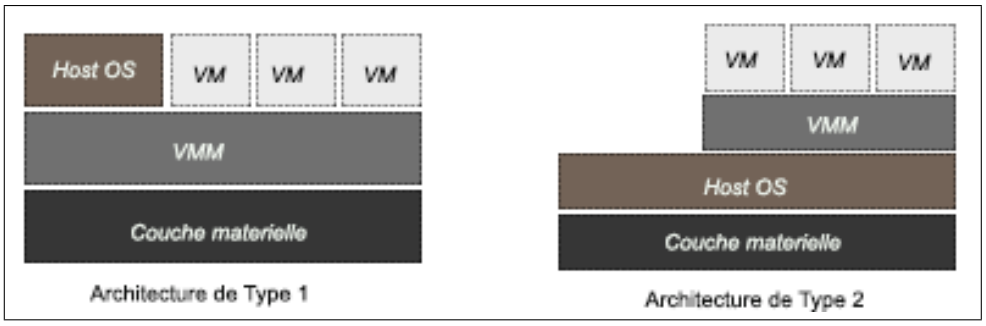

Figure 2.1 Différentes architectures d'hyperviseurs

2.2.3 Commutateur virtuel

Un commutateur virtuel (switch virtuel) est un logiciel qui permet aux machines virtuelles de communiquer entre elles. À chaque commutateur [1] virtuel, on associe un certain nombre d'adresses IP qui seront distribuées aux machines invitées grâce au service DHCP [2] (Dynamic Host Configuration Protocol).

Les commutateurs virtuels fonctionnent selon trois modes[vNe (2014)] différents mais, par

1. dans le domaine de l'infonuagique, un commutateur joue souvent le rôle de routeur virtuel
2. Protocole réseau qui assure la configuration automatique des paramètres IP d'une station.

défaut, le *mode NAT* (Network Address Translation) est utilisé. Dans ce mode, une machine virtuelle se sert de l'adresse IP de l'hôte pour communiquer avec les machines physiques extérieures. Cependant, celles-ci sont incapables d'initier une communication avec la machine virtuelle.
Pour le *mode routé*, le commutateur est directement connecté au réseau physique de l'hôte pour véhiculer les paquets. Un mécanisme permet d'inspecter les paquets pour les rediriger efficacement vers la destination. Les machines virtuelles appartiennent à un sous réseau virtuel invisible pour les autres systèmes physiques (ceux-ci ne peuvent pas les contacter). Il est nécessaire de configurer les routeurs physiques de la plateforme afin de permettre aux machines extérieures d'avoir accès à ce sous-réseau.
Dans le cas du *mode isolé*, les machines virtuelles connectées au commutateur peuvent communiquer entre elles et interagir avec leur hôte, mais la communication avec le monde extérieur est impossible. Ce mode peut être privilégié pour des raisons de sécurité dans certaines plateformes. Dans cette étude, nous utilisons comme commutateur virtuel Open vSwitch[Pfaff et al. (2015)], qui est un logiciel conçu pour supporter d'énormes flux réseau.

2.3 Types de services infonuagiques

Lorsque nous parlons de services infonuagiques, nous faisons en fait référence à trois modèles de services *(calcul, réseau et stockage)* qui fournissent chacun des ressources spécifiques. En règle générale, ces trois services sont déployés dans la même plateforme infonuagique pour garantir la disponibilité de plus de types de systèmes virtuels (espace de stockage, réseau virtuel ...) pour les utilisateurs.

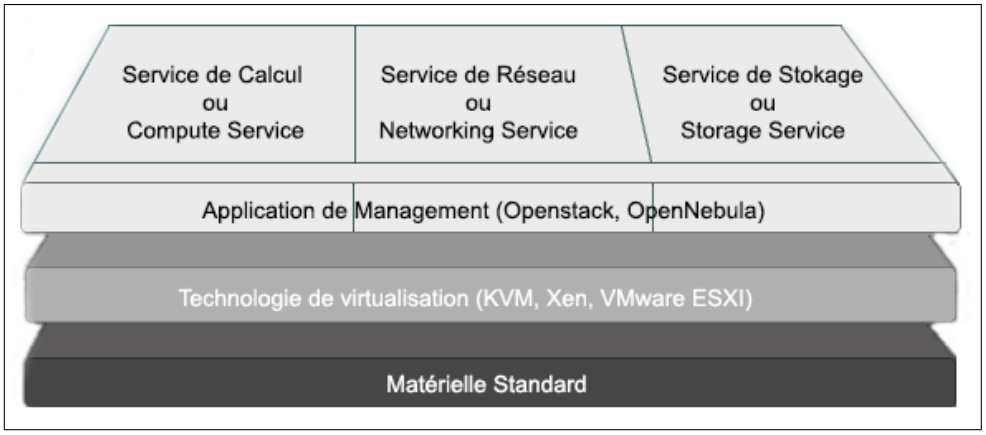

Figure 2.2 Architecture des services infonuagiques

2.3.1 Services de stockage

Ce service fournit des espaces disques pour le stockage des données des utilisateurs. Il est déployé selon deux modèles :

— *Usage direct par des utilisateurs* - L'usage d'un service de stockage est une solution alternative qui procure aux entreprises un moyen de conserver leurs informations en dehors de leurs propres infrastructures. En cas de perte ou de détérioration de leurs disques, elles pourront récupérer leurs données en ayant accès au service. Ce modèle décline un moyen simple de partage de données et un espace collaboratif pour les utilisateurs. Dropbox[Houston and Ferdowsi (2008)] est un exemple de ce type d'architecture.

— *Usage par des machines virtuelles* - Dans ce cas, il est basé sur la virtualisation des disques physiques et est utilisé pour donner aux machines virtuelles un environnement de stockage permanent. L'accès aux disques virtuels se fait souvent avec des interfaces traditionnelles telles que iSCSI ou NFS. Ces disques sont attachés et montés par les systèmes invités.

2.3.2 Services de réseaux

Les services de réseaux (networking service) offrent un ensemble de ressources virtuelles (routeur et commutateur virtuels) pour l'interconnexion des VMs. Ils permettent de réduire considérablement le temps de configuration des équipements, de définir des topologies de réseaux assez complexes (plus aisément qu'en utilisant du matériel physique) et autorisent les utilisateurs à établir des politiques de détection d'intrusion et de routage des paquets. Les services de réseaux sont souvent utilisés pour répondre à deux besoins :

— *Interconnexion des machines physiques* - Ce cas de figure permet de se départir d'équipements de réseaux pour la communication entre des systèmes physiques. Une machine physique exécute une application qui émule le fonctionnement d'un routeur pour acheminer les paquets.

— *Interconnexion des machines virtuelles* - Dans ce cas, l'objectif principal est de fournir aux systèmes invités présents dans un même hôte, un moyen d'interagir entre eux grâce aux commutateurs virtuels, ou de communiquer avec ceux déployés dans d'autres hôtes grâce aux routeurs virtuels.

2.3.3 Services de calcul

Dans ce modèle (computing service), les équipements physiques exécutent des applications de virtualisation (hyperviseurs) qui gèrent la création des machines virtuelles. Le fournisseur met à la disposition des clients, des interfaces pour contrôler leurs ressources virtuelles. Les entreprises ont recours à ce type de service afin d'avoir à leur disposition des unités de calcul pour effectuer des traitements à moindres frais. En effet, le service est facturé en fonction des ressources consommées. Comme ce service ne fournit pas d'espace de stockage, ni des interfaces de réseau aux systèmes invités, les VMs se retrouvent avec des fonctions limitées. Il faut alors déployer des services de réseau et de stockage pour disposer d'une infrastructure virtuelle qui sera plus proche d'un environnement physique.

2.4 Applications de gestion de services infonuagiques

Les applications de gestion *(dites Cloud Operating Systems)* de services sont utilisées pour superviser les machines et les technologies de virtualisation. Ces applications proposent des interfaces aux utilisateurs pour contrôler les infrastructures matérielles et virtuelles.

2.4.1 OpenNebula

OpenNebula[Milojičić et al. (2011)] est une solution de gestion des services infonuagiques. Il permet la mise en place de plateformes de type IaaS (Infrastructure as a Service). Il supporte les infrastructures composées de systèmes hétérogènes, et est compatible avec des hyperviseurs comme KVM, Xen ou VMware. L'évolution de OpenNebula est supervisée par la compagnie 'OpenNebula Systems' anciennement *C12G Labs*. OpenNebula offre une architecture flexible pour contrôler les tâches des machines dans le nuage. Cette architecture est composée de trois couches d'éléments.

— La couche des pilotes - Les éléments de cette couche communiquent directement avec le système hôte. Ils sont responsables de la création des ressources virtuelles et de l'allocation des espaces disques aux VMs.
— La couche centrale - Elle supervise le cycle de vie des ressources virtuelles. Elle a aussi pour rôle d'allouer les adresses IP aux VMs.
— La couche des interfaces - Cette couche est composée d'interfaces web et de consoles permettant aux utilisateurs de contrôler les systèmes physiques et virtuels.

2.4.2 OpenStack

OpenStack[Pepple (2011)] est né en juin 2010 de la collaboration de la NASA et de Rackspace. Depuis septembre 2012, son évolution est suivie par la fondation éponyme qui est supportée par une importante communauté composée de compagnies comme Canonical, Red Hat et Ericsson. OpenStack est une collection d'applications de supervision des ressources des infrastructures infonuagiques privées ou publiques. Ces applications permettent de gérer le matériel de réseau, de stockage et de calcul en fournissant des interfaces pour créer les ressources virtuelles selon la demande des utilisateurs. Il est déployé comme étant une infrastructure-service (Infrastructure as a Service) et est composé de plusieurs projets. Cependant, seulement six sont des projets principaux (Nova, Swift, Cinder, Neutron, Keystone, Glance). Les autres projets sont des projets optionnels, fournis par les membres de la communauté, ou des projets d'organismes tiers pour répondre à leurs propres besoins.

Les différents composants et services d'OpenStack communiquent à travers le service RabbitMQ[Videla and Williams (2012)] pour coordonner leurs actions. RabbitMQ est une application développée en Erlang [3] qui permet de gérer la communication entre plusieurs programmes clients et serveurs. Il utilise le protocole AMQP (Advanced Message Queuing Protocol)[Vinoski (2006)] pour l'échange des messages. L'utilisation du canal de messagerie pour la communication est délicate. En effet, si les messages arrivent de manière désorganisée (au niveau temporel), les services ne fonctionneront pas correctement. La mise en place d'un serveur NTP sur l'ensemble des hôtes est obligatoire afin qu'ils soient tous synchronisés sur la même date système. Il est possible d'entrevoir l'état de certains services et d'analyser leur fonctionnement grâce à la surveillance des communications qui passent par RabbitMQ. Nous proposons en Annexe [A], des résultats sur ce mode d'analyse.
Plus de douze versions différentes ont été présentées aux utilisateurs, mais notre étude est basée sur la version *Liberty* [4]. Toutefois, les résultats que nous exposons dans ce mémoire peuvent être obtenus avec les autres versions. Nous décrivons, dans la section suivante, trois projets que nous utilisons dans notre recherche.

3. Erlang est un langage de programmation utilisé pour construire des systèmes en temps réel souples, massivement évolutifs et avec des exigences sur la haute disponibilité.

4. Liberty (sortie en Octobre 2015) était la dernière version disponible au début de notre projet

2.5 Quelques services de OpenStack

2.5.1 Nova

Nova[Pepple (2011)] (OpenStack compute) est le principal service de OpenStack. Il est utilisé pour gérer les unités de traitement et superviser la création de machines virtuelles. Il est comparable à *Elastic Compute Cloud* d'Amazon. Son interface permet d'allouer des ressources sur la machine physique pour créer des VMs. Le service Nova est composé de plusieurs modules qui communiquent à travers un bus de message :

- API (nova-api) permet d'interroger et d'envoyer des commandes à l'ensemble des composants.
- Compute core possède trois éléments distincts. *nova-compute* gère l'interaction avec les hyperviseurs (QEMU/KVM, Xen, etc.). *nova-scheduler* sert à déterminer l'hôte où une machine virtuelle va être lancée. Il possède des filtres qui lui permettent de trier et de choisir les machines en fonction des caractéristiques des VMs à créer. Ainsi, il est possible de répartir efficacement les charges sur l'ensemble des hôtes. *nova-conductor* communique avec la base de données. Il est le seul composant à avoir accès à cette dernière pour éviter les problèmes d'incohérence liés à la modification parallèle de l'information.
- Le module réseau n'est activé que si on ne veut pas faire usage du projet Neutron. Ce module va interconnecter les VMs et contrôler le réseau grâce au composant *nova-network*. Ce dernier se charge aussi du routage et de la traduction des adresses. L'unité *nova-dhcpbridge* fournira des adresses IP et des ponts de connection pour la communication des VMs.
- Le module d'administration est fourni pour faciliter les tâches de supervision. Le composant *nova-client* permet aux utilisateurs d'envoyer des requêtes de réservation de ressources tandis que *nova-manage* est destiné aux administrateurs pour la gestion des instances.

2.5.2 Neutron

Neutron[Denton (2015)] est le projet de OpenStack qui permet de fournir un service de réseau et de s'assurer, en mettant en place des routeurs virtuels, que tous les systèmes déployés peuvent communiquer entre eux efficacement. Il est souvent utilisé avec le programme Open vSwitch qui simule le comportement d'un commutateur. Neutron est composé de quatre composants :

- Le module central (*neutron-server*) s'exécute sur l'unité de réseau. Il contrôle la com-

munication (bus de message) de Neutron avec le reste des services de la plateforme.
— Le module agent (*neutron-*agent*) est installé sur chaque unité de calcul (ou *compute node*) pour gérer la configuration du commutateur virtuel.
— Le module DHCP (*neutron-dhcp-agent*) est responsable de l'allocation des adresses IP aux VMs.
— Le module L3 (*neutron-l3-agent*) fournit un mécanisme de routage pour donner aux machines virtuelles un accès aux réseaux extérieurs.

Trois types d'entités sont gérés par Neutron. *L'entité réseau* : c'est le réseau virtuel créer par l'utilisateur. Ce dernier peut la configurer selon ses besoins. Plusieurs réseaux peuvent être déployés dans une même plateforme. *L'entité sous-réseau* représente un ensemble d'adresses IP qui seront assignées aux VMs du réseau. *L'entité port* est un port du commutateur virtuel. Les interfaces réseaux des instances [5] de VMs sont reliées aux ports.

2.5.3 Cinder

Cinder[Jackson et al. (2015)] est responsable du service de stockage. Il permet d'associer des espaces disques à des machines virtuelles. Il fournit un service semblable à celui qu'offre Amazon Elastic Block Storage. Cinder dispose de quatre modules.

— *cinder-api* est le point d'entrée du service de stockage. Il transmet les commandes au reste des composants.
— *cinder-volume* crée les disques virtuels et met à jour la base de données
— *cinder-scheduler* se charge du choix du nœud de stockage pour créer le volume
— *cinder-backup* permet de sauvegarder un volume de stockage.

Le service de Cinder déploie une solution iSCSI qui utilise le gestionnaire de volume logique LVM. Il est différent d'un service NFS, car chaque volume est associé à une seule instance de machine virtuelle.

Le Cinder gère trois types de ressources pour le stockage. Un *volume* correspond à un espace disque qui sera alloué à une instance de VM. Un *snapshot* est une copie d'un volume à un instant donné. Il peut être utilisé pour restaurer l'état d'un volume à une date précise si celui-ci est corrompu. Une ressource *Backup* est une copie archivée d'un volume qui est stocké dans OpenStack Swift.

5. une machine virtuelle est représenté par un objet au niveau de OpenStack

2.6 Outils d'analyse des plateformes infonuagiques

Il existe de nombreux outils pour la surveillance des opérations dans un nuage. Certains se basent sur la collecte et le traitement des logs des services[Bagnasco et al. (2015)], mais d'autres utilisent des sondes[Stallings (1998)] qui scrutent les systèmes matériels et les services pour connaitre leur état courant.

2.6.1 Logstash

Logstash[Turnbull (2013)] est un outil de collecte et de traitement des logs des applications. Il permet d'uniformiser les logs de différentes sources pour faciliter la détection de pannes. Il est aussi capable de récupérer le flux d'information circulant dans les bus de messages comme RabbitMQ[Videla and Williams (2012)] ou ZeroMQ[Hintjens (2013)]. Logstash possède un mécanisme qui filtre et transforme les données pour produire des traces selon le format désiré par l'administrateur. Après traitement, les informations sont stockées dans des fichiers pour une analyse ultérieure. Elles peuvent être envoyées vers le module Elasticsearch[Kuć and Rogoziński (2013)] qui va permettre la visualisation dans un programme externe comme Kibana[Reelsen (2014)] (interface web de visualisation de log).
Le mécanisme de traitement utilisé par Logstash est semblable à l'approche que nous utilisons pour intercepter les données de la couche application des plateformes.

2.6.2 Nagios

Nagios[Josephsen (2007)] fournit un service de monitorage des systèmes et des services informatiques. Il permet de surveiller l'état des machines et des réseaux dans un nuage grâce à des sondes appelées NRPE (Nagios Remote Plugin Executor). Les NRPE sont installés sur chaque hôte pour récolter les informations et les transférer au module central. Lorsqu'une anomalie est détectée, une alerte est générée à l'intention de l'administrateur. Nagios propose des interfaces graphiques qui présentent l'état des systèmes monitorés et des statistiques sur les logs des services. Son avantage est qu'il peut être utilisé sur différents types de plateformes infonuagiques (OpenStack, OpenNebula...). Grâce à son architecture modulaire, il est possible d'étendre son champ de surveillance en lui intégrant de nouvelles fonctions. Cependant, le mécanisme de sonde, qui lui permet à des intervalles réguliers (configurés par l'administrateur) d'aller interroger les services, n'est pas toujours efficace pour comprendre les pannes détectées. En effet, soit l'intervalle est trop grand et des changements d'état passent inaperçus, soit l'intervalle est assez petit, et dans ce cas, on se retrouve avec des surcoûts importants liés à l'usage de la bande passante.

2.6.3 Ceilometer

Ceilometer[Danjou (2012)] est un projet d'OpenStack qui fournit des modules de mesure de la consommation des ressources. Ce projet a pour but de simplifier la collecte des données et de centraliser la gestion des statistiques d'usage du réseau ou du CPU. Ceilometer est composé de trois services :

— La sonde (*polling agent*) conçue pour interroger aussi bien les services d'OpenStack que les systèmes physiques de la plateforme. Il utilise des protocoles comme SNMP [6] et IPMI [7] pour interroger l'hôte. Cette sonde est installée dans toutes les machines à surveiller.

— L'agent de notification (*notification agent*) écoute les messages qui passent sur le bus de notification (RabbitMQ). Ces messages sont convertis en évènements et en métriques pour la facturation des services offerts aux clients.

— L'agent de collection (*collector*) rassemble les données accumulées par la sonde et l'agent de notification, et les stocke dans une base de données.

— L'agent d'alerte évalue les informations et prévient les services externes si des conditions prédéfinies sont atteintes.

Ceilometer se limite à centraliser la collecte des données sans toutefois les traiter. Il est très souvent utilisé avec d'autres outils spécialisés dans la supervision des systèmes ou dans la facturation des ressources consommées.

2.6.4 Graphite

Graphite[Chau et al. (2008)] est une solution de surveillance qui permet la visualisation des métriques des services. Il peut être utilisé conjointement avec Ceilometer pour obtenir les statistiques d'utilisation des ressources et faire un diagnostic de l'infrastructure. Dans ce cas, Ceilometer collecte les données et les transfère à Graphite pour le traitement et la création des interfaces graphiques. Graphite est composé de trois éléments :

— *carbon* est responsable de la réception des métriques envoyées à Graphite.

— *whisper* est une simple base de données pour le stockage des métriques.

— *graphite-web* est une interface graphique qui permet aux utilisateurs d'interroger le service. Il est responsable de l'affichage des statistiques et de la génération des graphes.

6. Simple Network Management Protocol
7. Intelligent Platform Management Interface

2.7 Migration de machines virtuelles

Dans le contexte de l'infonuagique, la migration est un processus au cours duquel une machine virtuelle est déplacée d'un hôte à un autre. Elle consiste à transposer les ressources (mémoire, disque...) utilisées par un système invité vers une autre machine physique. Quand l'espace de stockage de la machine est présent sur un disque partagé par les deux hôtes impliqués dans le processus, la migration ne consiste plus qu'à déplacer la mémoire vive de la machine virtuelle.
De nombreuses situations justifient l'utilisation d'un tel mécanisme dans une plateforme infonuagique. En effet, lorsqu'on désire effectuer une maintenance sur une machine hôte, il est important de migrer les systèmes invités avant de la mettre hors tension, pour éviter la perte des données. En outre, la migration est nécessaire pour l'équilibrage de charge de travail sur les machines physiques, mais aussi pour répondre à des exigences écoénergétiques. Dans ce dernier cas, la migration est utilisée pour contrôler la consommation d'énergie de la plateforme en déplaçant les serveurs virtuels et en mettant hors tension les hôtes non utilisés. La migration se fait soit à froid[Buyya et al. (2010)] soit à chaud[Clark et al. (2005)] selon les exigences des services déployés dans les serveurs.

2.7.1 Migration à froid

Pour cette stratégie, le système invité est mis hors tension avant de copier l'état de la mémoire. Cela permet d'éviter les erreurs au cours de la copie, Toutefois, l'activité de la machine virtuelle est perturbée, puisque celle-ci est inaccessible pendant le processus. Arrivée sur l'hôte de destination, elle peut reprendre son activité dans le même état qu'avant la migration. Cette méthode est délaissée au profit de la stratégie à chaud, lorsque les applications du serveur virtuel requièrent une haute disponibilité.

2.7.2 Migration à chaud

Cette méthode nécessite l'utilisation d'un espace de stockage partagé entre l'hôte source et l'hôte de destination. Elle permet de transférer une machine virtuelle sans la mettre hors tension. Ainsi les applications restent disponibles pour les utilisateurs au cours du processus. La migration à chaud utilise souvent un transfert par *pré-copie*[Ma et al. (2010)] de la mémoire. Pour ce faire, la totalité de la mémoire est migrée vers la destination pendant la première phase. Mais, puisque le serveur virtuel est toujours actif, il se peut qu'il modifie certaines pages mémoires. Ces pages seront transférées au cours d'une seconde phase (phase itérative) jusqu'à ce qu'il ne reste plus de données remodifiées à copier.

Le transfert de la mémoire peut aussi se faire par *post-copie*[Hines et al. (2009)]. Dans cette méthode la machine virtuelle est démarrée à la destination et la mémoire est transférée au fur et à mesure que le système invité en fait la demande. L'état de la mémoire de l'invité est conservé sur la machine source jusqu'à la fin du transfert des données vers l'hôte de destination.

2.7.3 Conclusion

Les outils d'analyse des services infonuagiques, présentés dans ce chapitre, ne sont pas toujours efficaces pour localiser les anomalies des systèmes. Nous allons privilégier la technique de traçage, en utilisant LTTng, pour obtenir plus de détails sur les opérations des systèmes et comprendre les interactions des différents composants des services. Nous utilisons OpenStack pour mettre en place notre infrastructure. Trace Compass sera utilisé pour la visualisation et l'analyse des traces.

La migration de machines virtuelles est une technique très utilisée, mais des pannes peuvent survenir lors de l'opération, causant un ralentissement des services dans la plateforme[Bloch et al. (2014)]. Nous abordons plus en détail la migration à chaud dans le chapitre [4]. Dans le prochain chapitre, nous présentons la méthodologie de notre recherche pour parvenir à diagnostiquer les services de calcul, de réseaux et de stockage.

CHAPITRE 3 MÉTHODOLOGIE

3.1 Contexte de travail

Dans cette section, nous présentons notre environnement de travail et les caractéristiques des systèmes que nous utilisons. En plus, nous décrivons la procédure de mise en place de nos services infonuagiques.

3.1.1 Stations de travail

Nous avions à notre disposition trois machines (deux ordinateurs de bureau et un ordinateur portable) pour la mise en place de notre plateforme. Le tableau 3.1 présente les caractéristiques de celles-ci. Chaque machine exécute la version minimale du système d'exploitation Ubuntu 14.04 avec le noyau 4.2 : les applications inutiles pour notre recherche sont désinstallées pour réduire les interférences avec nos services.

Tableau 3.1 Caractéristiques des ordinateurs de la plateforme infonuagique

Type	Rôle	Caractéristiques	
Bureau	contrôleur	Processeur	intel i7-4790, 3.60GHz
		Coeur physique	8
		Mémoire vive	32GB
Bureau	compute	Processeur	intel i7-4770, 3.40GHz
		Coeur physique	8
		Mémoire vive	32GB
Portable	compute	Processeur	intel i7-4710HQ, 2.50GHz
		Cœur physique	8
		Mémoire vive	12Gb

3.1.2 Mise en place de la plateforme

Nous avons mis en place notre infrastructure en utilisant l'outil DevStack. DevStack est un ensemble de scripts qui facilite l'installation des services OpenStack. Nous utilisons la configuration 'Multi nodes' pour permettre le déploiement d'OpenStack sur un ensemble de machines. Ainsi, nous pouvons ajouter autant de machines que nécessaire, pour procéder aux différentes expérimentations.

Notre infrastructure est composée de trois machines (contrôleur, compute1, compute2) comme indiqué plus haut. Les machines compute1 et compute2 servent uniquement comme

machines hôtes pour le lancement des machines virtuelles. Le contrôleur exécute les autres services de gestion. Nous utilisons plusieurs services de sorte à reproduire un environnement de production pour notre étude.

1. *Horizon* fournit une interface web pour l'utilisation des services de OpenStack.
2. *Keystone* est un annuaire qui centralise les authentifications et les autorisations nécessaires aux services d'OpenStack.
3. *Glance* permet l'enregistrement et la distribution des images des machines virtuelles.
4. *RabbitMQ* fournit un mécanisme pour l'échange des messages entre les services de OpenStack.
5. *Nova* autorise la supervision des systèmes invités.
6. *Neutron* permet la mise en place de réseaux virtuels.
7. *Cinder* fournit un stockage persistant aux machines virtuelles.

Pour ce qui est de l'étude sur la migration, expliquée au chapitre [4], nous avons privilégié un transfert des données par le protocole SSH (Secure Shell) pour respecter les normes recommandées pour la sécurité des échanges.
Exemple de configuration minimale de DevStack pour le contrôleur et les computes
local.conf

```
[[local|localrc]]
Multi=1
enabled_services=g-api,g-reg,key
enabled_services+=n-api,n-cpu,n-sch,n-cond
enabled_services+=mysql,rabbit,dstat
enabled_services+=quantum,q-svc,q-agt,q-dhcp,q-l3,q-meta
live_migration_uri=qemu+ssh ://%s/system
```

3.2 Caractéristiques des systèmes infonuagiques

En rappel, notre problématique est que, pour mesurer les performances et diagnostiquer les anomalies des services infonuagiques, les traces noyau sont souvent insuffisantes, car elles ne définissent que des évènements (appels systèmes) élémentaires sans décrire les opérations sus-jacentes des services (création, destruction, migration de machines virtuelles ...). Ces opérations sont obtenues grâce aux traces utilisateurs. Il faut toutefois préciser que, lorsqu'elles sont utilisées toutes seules (sans traces noyaux), ces traces utilisateurs sont aussi insuffisantes pour un diagnostic efficace.

Notre méthodologie consiste à analyser les systèmes distribués sur plusieurs couches (noyaux et utilisateurs) et à faire ressortir les anomalies. Dans la suite de cette section, nous présentons les couches des machines dans un nuage et nous définissons les types de systèmes que nous analysons, avant d'exposer dans les détails le processus d'instrumentation des services.

3.2.1 Types de systèmes infonuagiques

Nous avons défini, dans le chapitre [2], les différents modèles de services infonuagiques (service de calcul, de réseau et de stockage). Ces services utilisent des machines avec des caractéristiques spécifiques. Nous pouvons identifier quatre types de machines physiques pour notre étude :

— Le *contrôleur* (ou *controller*) supervise et coordonne l'exécution des opérations des différents services.
— L'*unité de calcul* (ou *Compute node*) exécute les hyperviseurs (exemple : QEMU/KVM) pour la supervision des machines virtuelles.
— L'*unité de réseau* (ou *Network node*) exécute des applications (exemple : Open vSwitch) qui fournissent les ressources nécessaires pour la communication des systèmes physiques et virtuels.
— L'*unité de stockage* (ou *Storage node*) fournit l'espace disque nécessaire pour le stockage permanent des données des systèmes invités.

Chaque type d'unité dispose de plusieurs couches avec des spécificités différentes. Dans notre analyse, nous considérons qu'une machine peut être à la fois un contrôleur, une unité de calcul, une unité de réseau et une unité de stockage. Nous revenons sur ces types de machines dans le chapitre [4].

3.2.2 Couches des systèmes

Pour investiguer efficacement une machine, nous supposons qu'elle est subdivisée en trois couches : une *couche application de gestion*, une *couche de virtualisation*, une *couche noyau*. Le tableau 3.2 décrit les couches des unités. Plus de détails sont disponibles dans le chapitre [4]. Pour chaque unité, le diagnostic consiste à examiner la couche application, la couche de virtualisation et la couche noyau.

Tableau 3.2 Couches des unités des services infonuagiques

Couches	Unité de calcul	Unité de réseau	Unité de stockage
Application de gestion	Nova	Neutron	Cinder
Virtualisation	QEMU/KVM	Open vSwitch	-
Noyau	Noyau du système	Noyau du système	Noyau du système

3.3 Implémentation des points de traces

Dans cette section, nous présentons les principes d'instrumentation (ajout de points de traces) des couches des systèmes. Nous bénéficions déjà d'une couche noyau qui dispose de nombreux points de traces qui révèlent l'état des machines. Ici, nous nous intéressons à l'instrumentation des couches applications et des couches de virtualisation.

3.3.1 Instrumentation des couches applications

Notre étude est basée sur l'utilisation d'OpenStack, mais le principe de l'analyse en couche peut être étendu aux autres applications de gestion.
Les projets composant OpenStack sont écrits en Python, c'est pourquoi nous utilisons l'outil de traçage LTTng-UST avec l'extension ***Python-binding*** pour collecter les évènements au cours de l'exécution des tâches. Le traçage de la couche application (Nova, Neutron, Cinder) est basé sur les activités de *logging* des applications python pour récupérer tous les logs. Les logs disponibles dans OpenStack sont, pour la plupart, inutilisables pour une analyse efficace, car ils ne reportent pas assez d'informations pertinentes.
Pour répondre à nos objectifs, nous proposons de représenter les logs au format JSON pour standardiser les sorties de traces. Ainsi, chaque log ou évènement est représenté par un ensemble d'attributs définissant l'état du service. La figure [3.1] expose un exemple de sortie possible. Ce qui nous intéresse particulièrement, c'est l'état des instances de machines virtuelles pour le service Nova, des instances de réseaux pour Neutron et des instances de volumes (disques virtuels) pour Cinder. Nous avons écrit un ensemble de fonctions (disponibles ici [Bationo (a)]) qui retranscrivent les informations de ces instances.

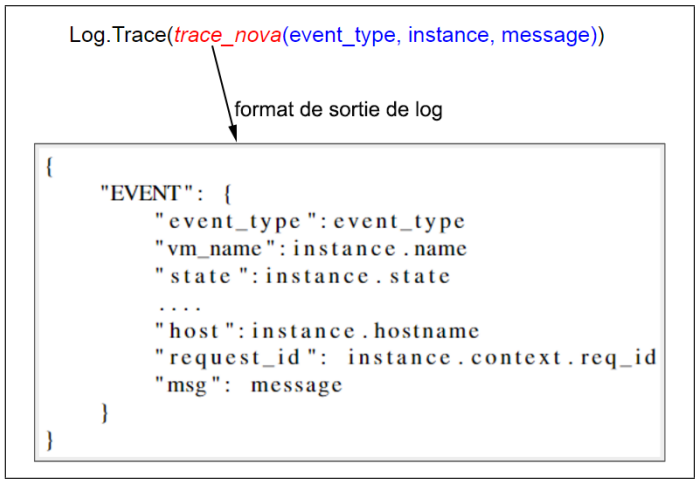

Figure 3.1 Exemple de sortie de log des couches applications

3.3.2 Instrumentation des couches de virtualisation

QEMU

En rappel, nous utilisons QEMU (version 2.5) pour superviser le déploiement des machines virtuelles. Il est largement instrumenté, et ces points de traces retranscrivent de façon détaillée, les interactions des systèmes invités avec l'environnement hôte. Cela est essentiel pour notre diagnostic des performances des services.
Même si QEMU dispose déjà de points de trace dans sa version officielle, pour procéder au traçage, il faudrait compiler son code source. En effet, par défaut, l'instrumentation est désactivée pour éviter un éventuel surcoût [1] lié au traçage.

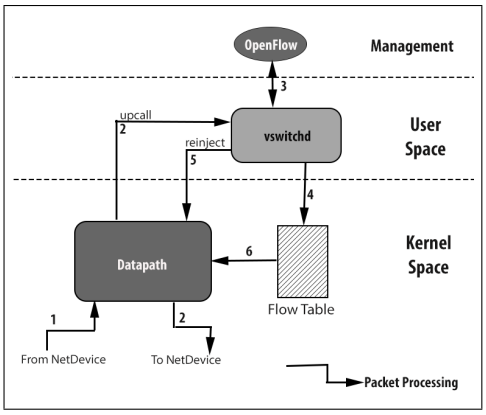

Figure 3.2 Processus de traitement des paquets par OvS

Figure 3.3 Structure d'un paquet réseau

1. Le traçage génère souvent des latences dans les infrastructures en production

Open vSwitch

Open vSwitch[Pfaff et al. (2009)] est un commutateur ou switch virtuel qui permet de router les paquets entre les machines invitées. Contrairement à QEMU, il ne dispose pas de points de traces dans sa version officielle [2]. Il est composé de deux éléments essentiels (ovs-vswitchd, datapath module) comme nous l'expliquons dans le chapitre [4]. Nous avons procédé à l'instrumentation du composant *datapath* qui gère les opérations de routage.
La figure 3.2 montre le chemin emprunté par les paquets dans le switch ; ce qui nous intéresse ici, c'est de retrouver ce chemin pour chaque paquet et estimer les performances (temps mis pour traiter un paquet) de chaque composant de Open vSwitch.

La méthode d'instrumentation adoptée est le marquage des paquets introduits par [Tarsel]. Elle consiste à ajouter un identifiant unique à chaque paquet dans sa queue (*tail room*, voir figure[3.3]). Nous avons utilisé l'instrumentation dynamique [3] grâce à *kprobes*. Kprobes est un mécanisme de débogage intégré dans le noyau Linux. Il permet d'insérer dynamiquement des points de trace dans le noyau au moment de l'exécution. Il s'agit d'associer un évènement à une adresse dans le code du noyau pour qu'à chaque fois que l'exécution arrive à cette adresse, un évènement soit généré.
Avec l'outil LTTng-addons, nous instrumentons dynamiquement les quatre fonctions principales responsables du routage de paquets dans le datapath. À chaque fonction, nous appliquons une opération spécifique pour retrouver l'identifiant du paquet traité. Le tableau [3.3] donne des précisions sur ces opérations. Ici, lorsque nous tombons sur une fonction enregistrée avec kprobes, deux cas sont possibles :

1. le paquet en cours de traitement ne dispose pas d'identifiant - alors, on lui ajoute un identifiant et on génère une trace pour LTTng. Ce cas correspond à l'étape 1 de la Figure [3.2]
2. le paquet dispose déjà d'un identifiant - on récupère l'identifiant et on génère la trace. Ce cas est associé aux étapes 2-7 de la Figure [3.2]

En ajoutant ou en récupérant les identifiants des paquets reçus, on arrive à estimer le temps de transfert vers la destination. Les fonctions d'ajout et de récupération sont présentées en [Bationo (b)].

2. La version 2.5 de OvS est la plus récente à ce jour
3. Les points de trace sont insérés au moment de l'exécution du programme. Il a l'avantage de ne causer de temps d'exécution supplémentaire que lorsque les points de trace sont activés.

Tableau 3.3 Instrumentation des fonctions de routages d'Open vSwitch

Fonctions	Point de trace	Actions
ovs_vport_receive	ovs_vport_receive	Ajouter Id au paquet
ovs_dp_upcall	ovs_upcall_start	Récupérer Id du paquet
ovs_packet_cmd_execute	ovs_upcall_end	Récupérer Id du paquet
ovs_vport_send	ovs_vport_send	Récupérer Id du paquet

3.4 Analyses des traces

Après l'instrumentation des couches des applications et de virtualisation, la prochaine étape est l'analyse des données obtenues par le traçage des systèmes. Nous cherchons à estimer les performances des opérations effectuées par les services et suivre l'exécution des différentes tâches, depuis la couche application jusqu'à la couche noyau. Nous considérons que chaque tâche lancée par les utilisateurs, depuis la couche application, est divisée en sous-opérations élémentaires dans les couches sous-jacentes (voir Figure [3.4]). L'analyse de la couche noyau est basée sur les outils disponibles dans Trace Compass à ce jour.

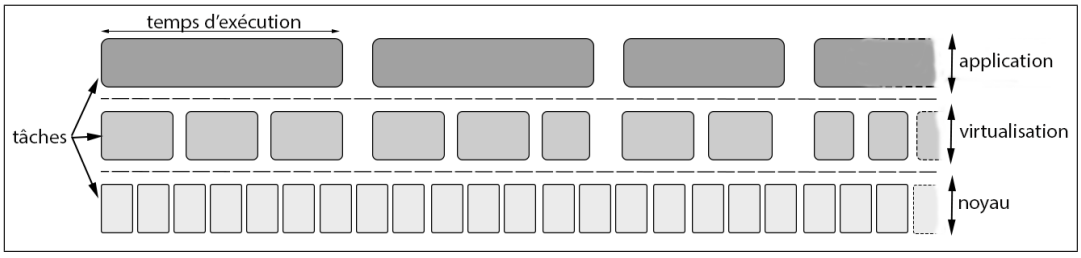

Figure 3.4 Analyse multicouche des services infonuagiques

3.4.1 Analyse de la couche application

Nous analysons les traces avec l'outil Trace Compass que nous avons introduit à la section [2]. Nous avons défini un certain nombre d'algorithmes pour générer des interfaces graphiques décrivant les opérations des services dans cette couche. Ces algorithmes sont disponibles ici [Bationo (c)]. La figure [3.5] présente un exemple d'évènement généré au niveau de la couche application. La date de création des évènements permet de mesurer la durée des tâches effectuées par les services d'OpenStack. Les données utiles pour notre analyse sont fournies par l'attribut *msg*. Il donne des informations comme le service concerné par la trace et le type d'évènement. Nous revenons sur l'analyse des services Nova dans la section [4.6], celle de Neutron est traitée dans [4.7] et celle de Cinder dans la section [4.8].

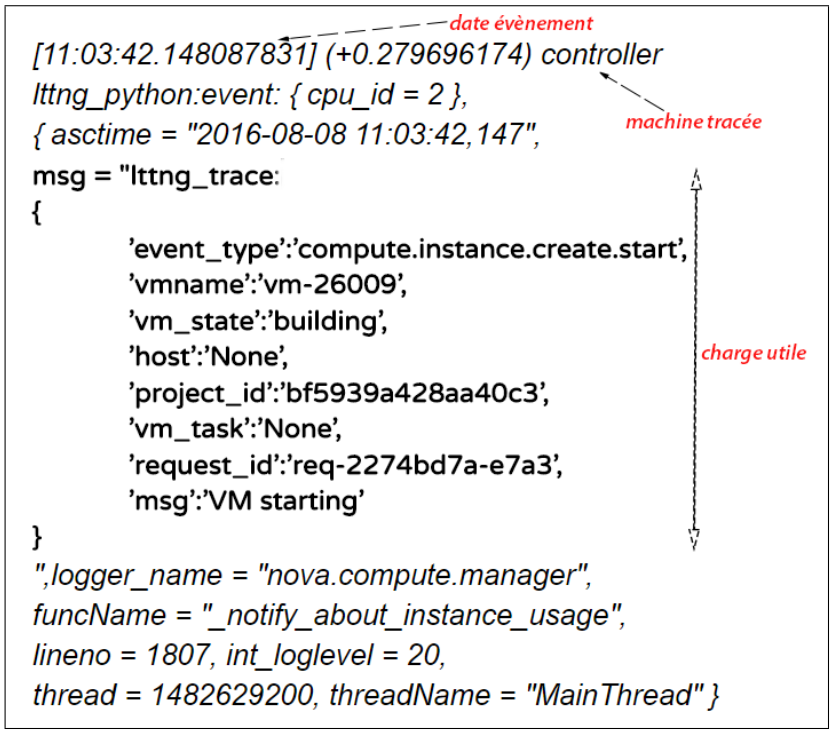

Figure 3.5 Exemple de trace Nova obtenu par LTTng

3.4.2 Analyse de la couche virtualisation

Avec les traces de la couche virtualisation, nous cherchons à avoir plus de détails concernant le déroulement des opérations de la couche application et à comprendre l'interaction des systèmes virtuels avec le système de l'hôte.

Analyse de QEMU

Concernant l'analyse des évènements de QEMU, notre intérêt s'est porté sur le processus de migration à chaud des machines virtuelles (voir section [2.7.2]). Le tableau [3.4] définit les plus importants points de trace pour déterminer le processus de transfert de la mémoire. La section [4.9] fournit plus de détails sur le diagnostic de la migration à chaud (live migration).

Tableau 3.4 Points de trace pour l'analyse de la migration à chaud

Points de trace	Définitions
migration_bitmap_sync_start	marque le début des étapes de migration de la mémoire
migration_throttle	réduction du temps d'accès de la VM aux CPUs
savevm_state_complete_precopy	fin du transfert de la mémoire ; début du stop&copy

Analyse de Open vSwitch

Les performances d'Open vSwitch (OvS) sont essentielles pour l'interaction des machines virtuelles. Aussi, lorsque le composant datapath reçoit un paquet, sa première action est de contacter ovs-vswitchd pour découvrir comment le traiter. Le résultat est stocké dans le cache pour les prochains paquets similaires. Le problème est que d'une part les opérations de ovs-vswitchd sont lentes et d'autre part la communication entre le datapath et le ovs-vswitchd engendre des coûts importants : cette communication doit donc être limitée au maximum.
Par l'analyse des traces de OvS, nous déterminons la fréquence de l'interaction [datapath/ovs-vswitchd] grâce à l'occurrence des évènements de type *ovs_upcall_start* [3.3]. Puis, nous mesurons la durée des opérations de ovs-vswitchd qui équivaut à la différence de temps entre les évènements de types ovs_upcall_end et ovs_upcall_start.

3.4.3 Analyse multicouche des services infonuagiques

L'analyse multicouche consiste à déterminer les tâches réalisées dans la couche application et à suivre leurs opérations sous-jacentes dans la couche virtualisation et noyau. Dans ce qui suit, nous désignons par *instance* un objet [4] décrivant, grâce à ses attributs, l'état d'un système donné.
Nous avons adopté les postulats suivants :

1. ***Chaque ressource virtuelle (ex : VM ou disque virtuel) est représentée, dans la couche application et dans la couche virtualisation, par une instance virtuelle de même nom qui retranscrit son état courant.***
   Exemple : instance vm-1000 représente la machine virtuelle de nom vm-1000
2. ***Chaque système virtuel est en fait un processus qui consomme les ressources (CPU, mémoire, disque) d'une machine physique.***
   Exemple : processus vm-1000 représente la machine virtuelle de nom vm-1000

Pour s'assurer de la validité des postulats précédents, il faut vérifier que Nova est toujours configuré avec l'option *instance_name_template="%(hostname)s"*, et QEMU avec l'option *set_procname=1*.
D'après le premier postulat, pour déterminer le comportement d'une ressource virtuelle, il suffit d'identifier l'instance correspondante (son nom est égal à celui de la ressource) dans les traces des couches application et virtualisation. Cependant, puisque ces données sont insuffisantes pour comprendre certaines anomalies de performance, on se réfère à l'analyse de la couche noyau pour obtenir des évènements plus détaillés.

4. Dans Openstack les objets sont définis en langage python

D'après le second postulat, pour diagnostiquer une ressource virtuelle, on va retrouver les données du processus qui lui est associé dans le système hôte. Ainsi, l'analyse multicouche des plateformes infonuagiques se fait à travers les associations :
$[\boldsymbol{INSTANCE}_{nom} - \boldsymbol{PROCESSUS}_{nom}]$.
$[\boldsymbol{SERVICE}_{nom} - \boldsymbol{PROCESSUS}_{nom}]$.

3.4.4 Analyse de traces de systèmes distribuées

Comme les plateformes infonuagiques sont composées de plusieurs systèmes physiques, nous collectons les traces de tous les hôtes vers une machine dédiée à l'analyse. Cependant, les traces collectées ne sont pas synchronisées même si, par défaut, le déploiement des services OpenStack requiert une synchronisation des machines avec le protocole NTP (Network Time Protocol). Dans notre cas, NTP n'est pas assez précis, puisque les évènements sont collectés à la nanoseconde près par LTTng. Nous allons ordonner les traces des différents systèmes grâce au mécanisme de synchronisation présent dans Trace Compass. Dès lors, nous pouvons suivre l'état d'une ressource virtuelle qui migre entre deux hôtes, et comprendre l'interaction des services situés sur différentes machines.
Dans la section suivante, nous présentons l'analyse multicouche des services de calcul, de réseau et de stockage des infrastructures utilisant OpenStack comme application de gestion.

CHAPITRE 4 ARTICLE 1 : CLOUD SERVICE PERFORMANCE DIAGNOSIS USING MULTI-LAYER TRACING

Authors

Yves J. Bationo
Ecole Polytechnique Montreal
yves-junior.bationo@polymtl.ca

Michel Dagenais
Ecole Polytechnique Montreal
michel.dagenais@polymtl.ca

Index terms - Cloud, OpenStack, Open vSwitch, QEMU, LTTng, Tracing, Performance analysis.

Submitted to Journal of Cloud Computing : Advances, Systems and Applications

4.1 abstract

Cloud services performance analysis is a real challenge, with the complexity of virtualization technologies. Issues can occur at any layer in the system, thus knowing the source of the deterioration is a first target for troubleshooting. In this paper, we introduce multi-layer tracing for cloud services, and define some parameters to consider for cloud infrastructure analysis. Our approach involves collecting data about cloud tasks from user-space and kernel-space of any cloud nodes. We then correlate the traces and show abnormal activities and failures in the system. Experimental results about virtual machines migration, a particularly difficult scenario to trace, confirm that we are able to diagnose service efficiency problems, locating a platform's weakest links.

4.2 Introduction

Cloud computing has recently emerged as an essential technology for sharing computer resources over networks. Cloud services refer to a number of resource management tasks like computing services, networking services and storage services. These services are usually provided in a single utility, based on cloud operating systems like Openstack. OpenStack is a large collection of projects that interact to handle users requests. The increased sophistication and complexity of cloud systems makes it difficult to detect service performance deterioration causes.

Tracing produces a large amount of data which describe device behaviours and provide precise information about resources state changes, and allow observing the delay occurring between any request and the corresponding response. Services in the cloud can be partitioned into 3 layers : application, virtualization and kernel, and some issues may occur at any of these. Locating the source of issues is the best first step to understand performance latencies. The idea is to trace and analyse each layer separately and correlate data output to pinpoint the source of failures.

We propose a tracing-based framework that helps system administrators to troubleshoot cloud services. Four main contributions are presented in this paper. First, we propose an approach for cloud nodes multi-layer tracing, with some identifiers that help to follow user requests and their execution through the cloud platform. Secondly we introduced OpenStack projects and services instrumention, for tasks latency diagnosis. Thirdly, we propose troubleshooting the compute virtualization layer through QEMU features tracing. The last contribution is about Open vSwitch instrumentation and performance diagnosis, while taking into consideration the time to switch packets.

This paper is organized as follows : section[4.3] describes related work and section[4.4] surveys cloud tracing background. In section[4.5], we present a multi-layer tracing taxonomy, and in section[4.6] we explain cloud computing services analysis. Section[4.7] studies the networking services performance, while section[4.8] displays storage services diagnosis features. Finally we discuss virtual machine live migration in section[4.9].

4.3 Related Work

Increasing interest in virtualization technology has led to the development of many tools for cloud system monitoring and troubleshooting, such as OSprofiler[Kumar and Shelley (2015)] for cloud applications, written in python. OSprofiler is a python library dedicated for application tracing. It is used to collect Openstack service events and functions calls. The recorded logs are used to build a call-tree showing the request flow and giving an estimation of the task performance. This feature is not suitable when the latency is due to some issues in the host operating system.

Marquezan and al.[Marquezan et al. (2014)] deal with the problem of identifying which management actions need to be performed in the cloud during a specific task. They proposed to display recorded trace events in a three-dimensional data space, instead of looking at them as isolated reports. They provided a framework which displays some diagrams that reveal the situation leading to some conflicting management decisions. The concept does not define how data is collected, but rather proposes a way to analyse these as a single block, by defining some metrics for the virtualization layer and the physical one. Information is collected from different layers of the computing node to reach more efficient decisions. Their work is limited to displaying what actions are appropriate for the cloud owner but, in this paper, we plan to show how to dig into cloud systems to get detailed reports, for troubleshooting and performance analysis, using LTTng. Following this concept, Montes and al.[Montes et al. (2013)] introduced the idea of monitoring levels in a virtualised environment. They proposed a management taxonomy and a tool called GMonE which covers the analysis of all the cloud components, as much as possible. Their work defines, according to the platform infrastructure, different interfaces for both providers and consumers to fulfill their management decision and to guarantee the QoS specified in the service level agreement. This tool can be used to find out the sources of some issues caused by system misconfiguration.

Waller and al.[van Hoorn et al. (2012)] presented Kieker, a framework for distributed software, which provides comprehensive data about the monitored system internal behaviour, and supports live analysis of the software traces. It provides measurement probes for performance monitoring and control-flow tracing. Some analysis plugins extract architectural models and visualize them as calling-dependency graphs and sequence diagrams. Kieker can have a large impact on the performance of the platform and it does not produce precise data about system-level measures, such as CPU and memory usage, like our framework does with LTTng.

Mi and al.[Mi et al. (2012)] proposed a way to locate performance issues with application

logs. They record information, during the execution of requests, to retrieve the behaviour of the worker. From any request which induces a service degradation, they aim to pick up the abnormal function. This method is extended by Ruben[Acuña et al.], which addresses the problem of analysing task workflows and service orchestration in the cloud. In his approach, python applications are instrumented by adding some code to be executed during task processing. He presents data-flow views, illustrating service dependencies based on file calls. This method is not sufficient to represent the cloud infrastructure ; the collected data is insufficient because the work is focused on the management software written in python and not on the whole system.

The work of Gebai[Gebai et al. (2014)] is closer to our research as he studied virtual machines and hosts traces synchronisation. He showed how to retrieve systems state and analyse virtualization technologies performance. Moreover, with his framework, he was able to detect competing threads in distinct virtual machines. Here, we apply those results for kernel traces analysis to get fine grained information about cloud platforms.

4.4 Background

4.4.1 LTTng

In this paper, we use LTTng[Desnoyers (2009)] to get cloud devices and services behaviours. LTTng is an open source tracing framework for Linux which provides a unified trace of events and adds a minimal overhead on the system. It offers features for remote tracing, collecting the trace from one host to a specified node. It is able to gather data from host's user and kernel space. For tracing the Openstack project, we use the 'LTTng-UST python bindings' to record the logs. Furthermore, we can get all the official tracepoint events of Linux kernel. In this work, we define some new tracepoints, to get detailed information about the studied nodes and services behaviour.

4.4.2 Trace Compass

Trace Compass[tra (2015)] is an open source framework for viewing logs and traces. It produces graphs for applications behaviour analyses. TraceCompass uses a State History Tree (SHT) [Montplaisir-Gonçalves et al.] which is a disk-based structure to handle large streaming interval data. The SHT provides an efficient way to store interval data on permanent storage with fast access time. Here, this data is obtained from state intervals computed from cloud system events, traced with LTTng. To analyse cloud systems, we have to be sure that

all the traces from the different nodes in the environment are synchronized. In fact the time interval taken by the platform to perform critical operations is important for administrators, to verify that service interactions were made as expected, in order to figure out executions faults. The time to complete an operation depends on the location of the service. When workers are located in different hosts, network congestion can cause tasks latencies. It is important to understand that, in the case of local communication, shared CPU congestion will lead to the same effect. Thus, we want to ensure that no service is blocked unnecessarily by another one.

Time synchronization is usually based on the Network Time Protocol (NTP), to allow all the cloud devices to have the same time reference, complete the activity in an orderly way, and retrieve the traces with timestamps based on this common reference. However, NTP is insufficient to guarantee good accuracy, as explained by Giraldeau and al[Giraldeau and Dagenais (2016)]. Trace Compass handles this matter by offering a trace synchronization feature based on their work. Synchronisation will be handled here by a convex hull algorithm implemented using kernel events (inet_sock_in/inet_sock_out) related to network communications between the compute nodes. Therefore, the synchronization of the clocks in the different nodes is not changed and does not need to be accurate. The traces collected from the different nodes are accurately time synchronized at analysis time, using the information from the events about exchanged network packets.

In this work, we define a certain number of attributes in the state tree of Trace Compass, to be used for graphs and views related to Cloud services. The needed attributes define the "state provider" [Montplaisir-Gonçalves et al.], that relates the behaviour of the resource states to the events reported in the trace collection.

4.5 Multilayer Tracing

In this section, we introduce the multi-layers tracing taxonomy and classify cloud hosts and services.

4.5.1 Nodes structure overview

We consider that any cloud node is composed of mainly three layers which cooperate to handle users tasks. Thus, analysing the performance of a cloud installation requires to coordinate the traces from these layers.

— *Application layer* : the service management interface located at user-space level. This layer handles customer requests, manages virtual devices life cycle, controls the hypervisor and ensures load balacing[Li (2014)] among the hosts. This interface acts like a cloud operating system and provides a large number of independent services for computing, networking and storage hosts supervision. A collection of API helps getting or sending data to the virtual device (virtual machine, NIC or volume) monitor. The application layer interface does not have direct access to any virtual device but provides logical objects : a "virtual instance" which describes the state and the task of the related device. This instance parameters are updated by polling the virtualization layer to get the current state and behaviour of the instance. The cloud operating system considered here is OpenStack. We use Nova for computing services, Neutron for networking and Cinder for storage.

— *Virtualization Layer* : the second layer on the host. Its role is to virtualize physical resources used by guests, independently of their operating system. More information about virtual devices can be found through that layer. It is responsible for guests OS translation to the hardware, in the case of computing services, and packets switching in the networking service. Most often, the emulation runs slower than the physical hardware. As a consequence, performance of virtual instances will decrease. The virtualization layer is usually associated with software systems like QEMU, KVM or Xen (for computing services), and Open vSwitch (for networking services). Tracing these tools helps detect virtualization technologies latencies.

— *Host kernel layer* : the kernel space. From the kernel side, any virtual device is seen like a regular process using computing resources. It is then easier for us to get fine-grained data about usage statistics, and measure the hypervisor consumption on behalf of each virtual host, to figure out its overhead, as introduced by Ankit and al.in [Anand et al. (2012)]. From this layer, we also get physical resources sharing and contention information and deduce virtualization features performance.

4.5.2 Nodes and Services

We have three types of cloud services : computing services for computing resources (CPUs sharing), networking services when we deploy technologies like NFV (Network Function Virtualization), and storage services for disk space sharing. These services are often deployed within the same platform, to grant more resources for consumers, but they require some specific hosts configuration. We have four kinds of hosts handling cloud activities.

— *Controller node* : it is a key component which manages the system environment, retrieves user tasks, and initiates load balancing mechanisms in the platform. For this host, only the cloud operating system needs to be traced.

— *Compute node* : it represents the host where virtual machines are running. It is the main node pulled by monitoring agents to collect virtual devices state and resource usage. Cloud computing services are deployed in this host. For any node of this kind, the three layers are traced.

— *Network node* : it provides data related to IP address assignments, network functions configuration, and the virtual network communications. Here, the tracing is focused on the application layer and the kernel space, to get the networking system behaviour.

— *Storage node* : is responsible for delivering persistent storage for virtual machines, it handles cloud storage services. It displays disk space shared between virtual machines.

A host in the cloud can be at the same time a controller, a compute node, a network node and a storage node. In this case, the host type will depend on the cloud service we are interested in.

	Layers \ Nodes	Controller	Compute	Network	Storage
Computing Service	*Application*	✓	✓		
	Virtualization		✓		
	Kernel	✓	✓		
Networking Service	*Application*	✓		✓	
	Virtualization			✓	
	Kernel	✓		✓	
Storage Service	*Application*	✓			✓
	Virtualization				
	Kernel	✓			✓

Figure 4.1 Cloud Services Node Layers Tracing

4.5.3 Cloud trace analysis

Monitoring and troubleshooting cloud systems requires splitting any issue in smaller problems from different components. The idea is to trace nodes and gather the data in one host using LTTng [Desnoyers (2009)]. Our approach consists in tracing user-space applications and kernel space processes to detect irregular situations. User-space applications run with normal privileges but kernel space threads implement system calls in the host. Information from these spaces will be correlated to understand the cloud environment and explain performance degradations. We consider that any cloud node is divided in 3 logical layers as explained above.

Having a complete view of the infrastructure is difficult because tracing must be done in different layers. We get the entire picture of the node by combining the traces from these layers. Depending on the cloud service analysed, we are collecting traces from specific layers (more details are available in Figure-4.1). The controller node is traced for all kinds of service diagnosis. However, we trace compute nodes for computing services, network nodes for networking services, and storage nodes for storage services analysis. The traces are imported in TraceCompass to be synchronized and produce graphs for performance analysis, as shown in Figure-4.2. In this paper, all the traces collected for the experiments are available in [Bationo (c)]. For computing service analysis, we associate each Nova instance to a QEMU instance and a kernel process (Figure-4.3), based on their name in the cloud environment, and we thus get all the required data for the virtual machine behaviour investigation. Table-4.1 displays the main parameters required to correlate the data collected from these nodes.

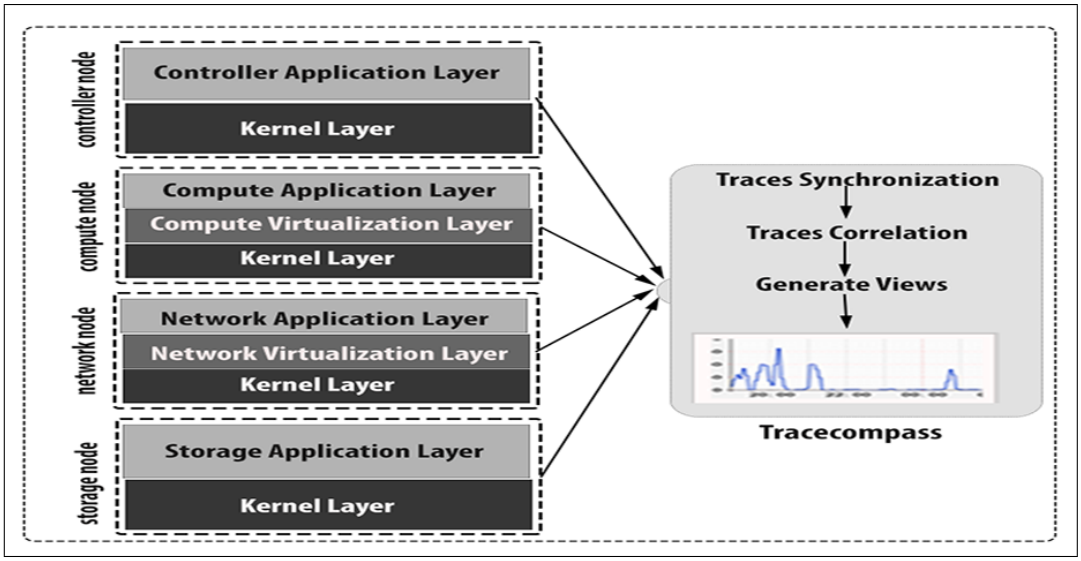

Figure 4.2 Cloud Infrastructure Tracing

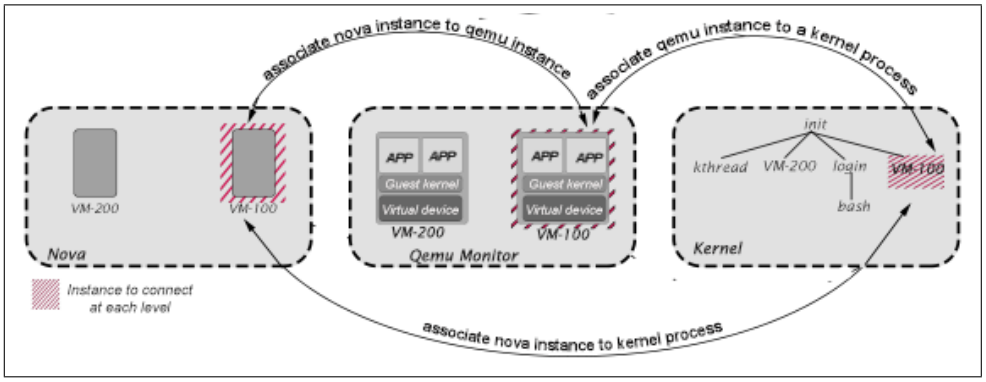

Figure 4.3 VM machine mapping : correlate vm100 data from 3 layers

Table 4.1 Required Parameters to correlate nodes data

Parameters	**Description**
$USER_{id}$	represents the tenant identifier that indicates which user made the request from the high-level interface. It is important to identify the user because, depending on its privilege, some task may fail if it does not have the correct credentials. These kinds of faults are not relevant and must be ignored during the diagnosis.
$PROJECT_{id}$	is defined by the provider. A user can be associated with many projects.
$REQUEST_{id}$	describes the request identifier made by any user of a specified project. With this parameter, we can link all the services which handle the related request and report about the task work flow.
VM_{name}	is the virtual machine display name or virtual machine instance name from the Nova event. Each instance name is related to a process with the same name in the compute node.
$SERVICE_{name}$	defines the OpenStack service which takes part in the task. This parameter corresponds to a process with the same name in the low-level trace event.
$TASK_{name}$	describes the activity performed, like VM creation, migration or network assignment. In most cases, it derives from the Nova trace event type.
$PROCESS_{name}$	represents a process on the compute OS. Each process is related to a virtual machine or a service. With logical objects from Nova, we can derive their associated entity in the kernel and follow the entity behaviour from higher to lower layers.

4.6 Computing Service Diagnosis

Here, we focus our work on virtual machine activities. Any task related to the VM state will be tracked down from the computing management interface to the virtualization layer and the kernel host, because observing the behaviour of all processes in the host is the best way to understand the current performance of the infrastructure. Table-4.1 displays the main parameters required to correlate the data collected from cloud nodes, considering these three layers. We use Nova as the high level layer and QEMU as the virtualization layer.

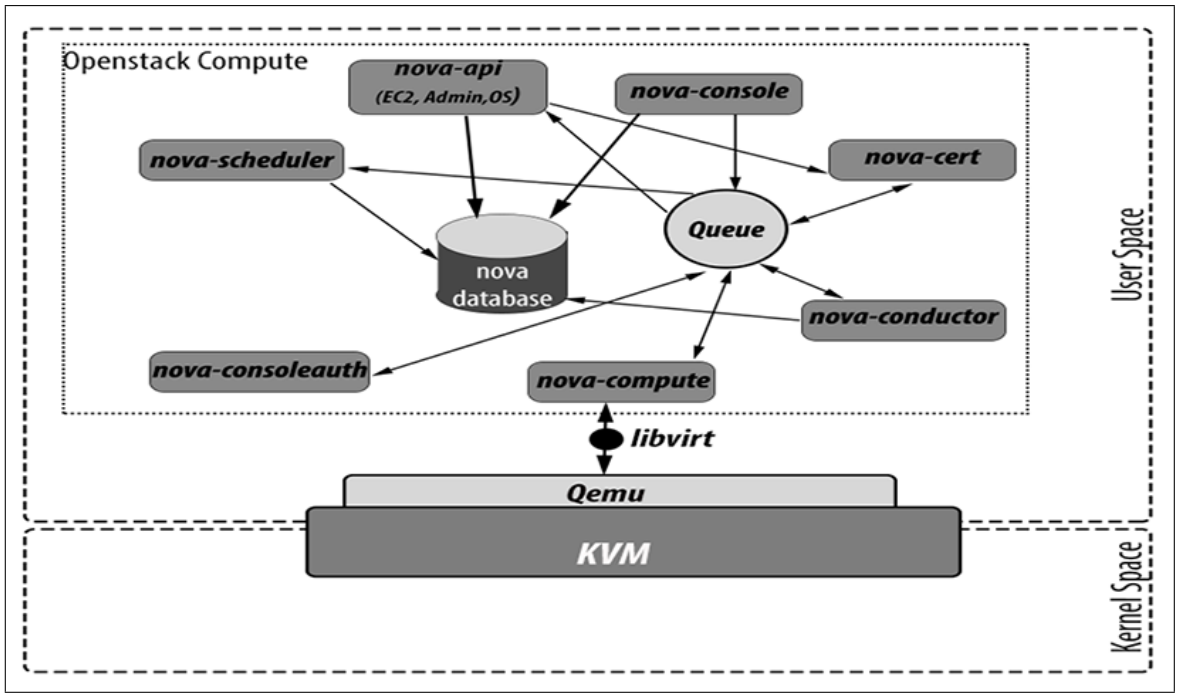

Figure 4.4 Compute Node Architecture

4.6.1 Nova Analysis

Nova is an Openstack project used for managing cloud computing systems. It is composed of many services that interact with each other to handle users requests. However, that make troubleshooting more difficult, as we must investigate more log files to pinpoint causes of failure. It is one thing to detect an error, it is another thing to understand the factor leading to that issue. Nonetheless, requests from, and responses to, guests are essential to locate any deficiency by recovering VMs states and tasks repartition among computing devices. To report about task disruptions, we proceed with the instrumentation of methods related to the virtual machine life cycle. We based our work on Nova logging activity by inserting

tracepoints similar to log entries. Thereby, the tracing process benefits from current logs describing the system component states. The trace output is similar to the log format but is more structured and standardised, to make the output investigation easier. We provide the most interesting information, which will help for monitoring and debugging. In our case, we choose to write the traces in JSON format, and retrieve values about event type and instance objects, which define the attributes of virtual machines. Table-4.2 displays a short list of events prescribed for Nova analysis ; more events and the tracing requirements may be found in [Bationo (c)]. Usually, in a cloud environment, we have more nodes running Nova services, we thus must collect traces from all of them, and synchronize their traces.

TRACE OUTPUT :

```
{
    "EVENT": {
        "event_type":event_type
        ....
        "device":device_name
        "request_id": context.request_id
        "message": msg
    }
}
```

Table 4.2 Nova events [short list]

sub-services	events
nova-compute	instance.create.{start/end}
nova-compute	instance.power_off.{start/end}
nova-scheduler	scheduler.select_destinations.{start/end}
nova-conductor	provider_fw_rule.{start/end}
nova-driver	libvirt.find.migration.min.data.{start/end}
nova-network	allocate.network.instance.{start/end}
nova-network	lease_fixed_ip.instance.{start/end}

Nova activities views are based on the attribute tree defined in Figure-4.5. Each attribute node represents a system resource. In this model, we consider 'VM State' to represent the current condition of the virtual machine and the 'Service Task' state to describe the service activity. Our analysis views give a detailed illustration of the cloud high-level infrastructure and serve many goals.

— *VM state investigation* : we mainly focus our audit on virtual machines, services and physical resources. As the user requests sollicit the virtual machine, knowing the cur-

rent state of the guest is a good way to deduce service failures when the instance state does not change as expected. The instance can take many states as defined in Table-4.3 according to the activity executed on the guest through the nova-compute service. The VM state view (Figure-4.6) shows a timeline of the state for each instance in the cloud system.

— *Services performance analysis* : We investigate functions processing by providing some performance measurements : the time to execute an operation. With the service view (as shown Figure-4.7) we estimate Nova main tasks efficiency, like the scheduling process, to show the performance of filter algorithms which select the compute node where the guest will be launched. We are able to check load balancing among services by detecting inactive workers.

— *Survey request flow* : Network congestion or messaging hub bottlenecks will reduce task efficiency. We have to check the activity process to ensure that no service is waiting unnecessarily for another, and verify that the request is handled as expected by the workers.

— *Troubleshooting* : Nova components generate logs (Figure-4.7) that depend on the fault severity, like debug, warning, error. For each critical log, we pinpoint the request flow leading to that fault by identifying the sensitive operation. Then, we can identify the critical function and start debugging.

— *Resource consumption* : The quantity of available computing resources in the traced environment gives us a good appreciation of service behaviours and platform resource usage. We fully instrumented the Nova compute-manager to get a precise report about resource usage in such a way that we get data for creation, update or deletion of any virtual device. Based on this information, we provided graphs about virtual machines repartition per host, VCPU usage, and ram reservations.

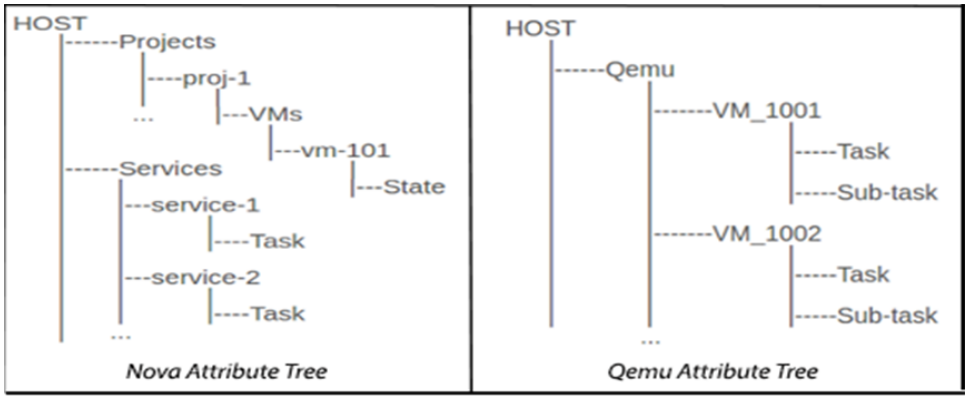

Figure 4.5 Computing Diagnosis Attribute Trees for TraceCompass

Table 4.3 Nova instance states

VM State	Description
BUILDING	It proves that the VM creation has started; block device and volume are mapped to the VM instance and the network IP is assigned.
ACTIVE	Shows that any operation on the VM lifecycle is complete, the VM is fully running.
STOPPED	the related virtual machine is not using the computing resource anymore, all the memory is stored in physical storage.
RESIZED	VM is stopped and is not running on the source node but is running on the destination node. VM is located on two different hosts at the same time, waiting for a commit.
MIGRATING	VM is migrating from one host to another, this state is different from resize because the VM at the source is not stopped during the task.
ERROR	Something goes wrong during the operation, VM state is unrecoverable.

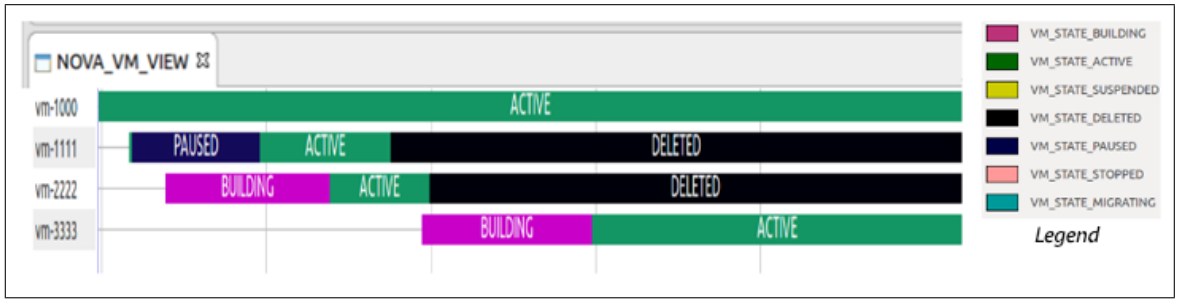

Figure 4.6 Nova Instance State Changes

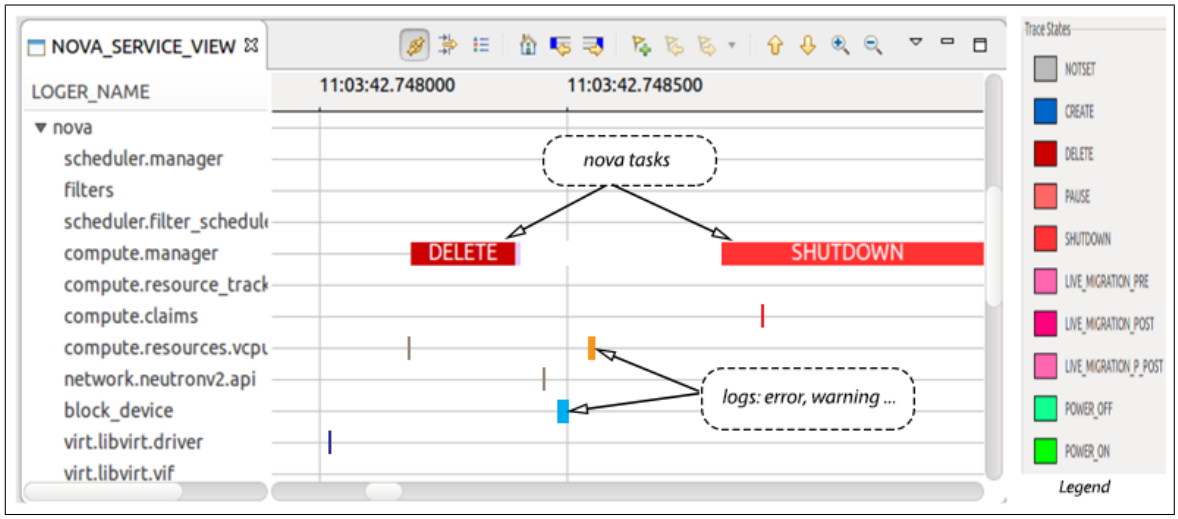

Figure 4.7 Nova Services log output

4.6.2 QEMU Analysis

In this paper, the computing service virtualization layer (known as the hypervisor) is QEMU-KVM. We use QEMU[Bellard (2005)] on top of KVM[Kivity et al. (2007)]. QEMU is a hardware device emulator which provides a virtual interface for physical disks and network resources usage. KVM is included in the Linux kernel and uses hardware extensions to accelerate virtualization and supports guests running unmodified OS images.

This level performs memory abstraction and achieves the mapping of the guest memory to the physical memory. In such environments, the guest does not have direct access to the host memory. On top of that, the virtualization layer provides, with the help of the CPU abstraction mechanism, some VCPUs (virtual CPU) for each VM. That process requires to trap critical operations in the VMM so that the resource access would be redirected through the host, to guarantee the stability of the system. Nevertheless, non-sensitive instructions of the VM are run directly to improve the performance. Otherwise, it also handles the multiplexing of I/O request operations of the virtual machine on the physical host. By tracing QEMU with LTTng, we get all these internal activities and find the memory leaks, or the memory access contentions between virtual machines, and figure out how QEMU handles their requests to the hardware. We analyse the middle layer information of the VM using QEMU events, following the attribute tree described by Figure-4.5. We identify a virtual machine main activity and their sub-tasks which provide more details about their current behaviour. The operation provided by QEMU can be : VM creation, live migration or VM destruction. In this work, we use the current tracepoints in the QEMU application. We contribute by defining some metrics and algorithms for TraceCompass to generate graphs and views. These algorithms and the list of events used to monitor guest migration are available in[Bationo (c)].

4.6.3 Kernel Analysis

In this section, we address low-layer tracing using LTTng for physical devices investigation. Giraldeau and Dagenais[Giraldeau and Dagenais (2016)] show how to understand service slowdown by analysing kernel traces to find the source of bottlenecks. They are able to pinpoint any latency due to network or CPU contention. This work can be extended to OpenStack Nova workers which interact to handle consumer requests. After knowing which service is concerned with an issue from higher layers, we can explain the task performances with kernel trace investigations. Thus, any service which waits unnecessarily for another can be detected, if we know its workflow. Both services and virtual machines are viewed from kernel traces as processes sharing resources. That is why we can analyse these as regular processes, which use computing resources, and study their performance based on system metrics,

as introduced by Desnoyers[Desnoyers (2009)] and Giraldeau[Giraldeau et al. (2011)]. Processes in the host compete with each other to use the CPU and, when one gets the CPU for a long time, it impacts the execution time of other processes and their performance, because they stayed in the blocked state, waiting for their turn. That mechanism causes resource sharing interference, which impacts negatively the performance of virtual machine applications. We consider three main factors when we investigate kernel processes efficiency :
CPU consumption - $\text{wait}_{\text{time}}$: total time the process was waiting for the CPU. A process with high priority monopolised the CPU for such a long time that other processes have to wait until it finished its task. $\text{CPUs}_{\text{usage}}$: is the time during which the process uses the CPU, it will give an idea of how much the virtual machine is consuming within the system.
Memory usage - gives the memory usage per service or guest process and provides more accurate data about memory leaked when a virtual host is using memory pages without releasing them.
I/O access statistics - provides the I/O operation latency, network and disk access. A lot of features for resources viewing are available in the LTTng and TraceCompass tools to analyse the kernel of the host and get fine-grained information to explain system usage, and show service degradation.

4.7 Networking Service Diagnosis

In this section, we describe networking service component behaviours and analyse some common infrastructure issues. Cloud networking services have to deal with many threats that decrease the quality of service. A major problem usually encountered in the networking platform is "Packet transmission high latency" which slows down the interactions between virtual machines and the complete activity of cloud environment consumers. That issue is generally caused by factors like :

— *Network congestion* which has two dimensions : Physical network congestion is the result of extra bandwidth consumption by the host in the cloud, due to a high network load. When communication matters for consumer tasks performance, its recommended to launch related virtual nodes (with which they interact frequently) in the same physical host to reduce usage of the physical network bandwidth, and increase the guest interaction quality. Another kind of congestion is virtualization layer congestion due to packet switching latency. With a high latency virtual switch in a cloud infrastructure, packet transmission will take too much time, as the vSwitch does not have enough capacity to handle them.
— *CPU Congestion* Increasing the networking nodes workload will slows down the vS-

witch software and decreases its performance. In fact, the more guests we have, the more the vSwitch has to compete with them to use the CPU, to be able to handle the packets they send and receive. That case will lead to an undesirable scenario which impacts the global performance of the platform.

Our networking service infrastructure consists in OpenStack Neutron (high-level layer) over Open vSwitch (virtualization layer).

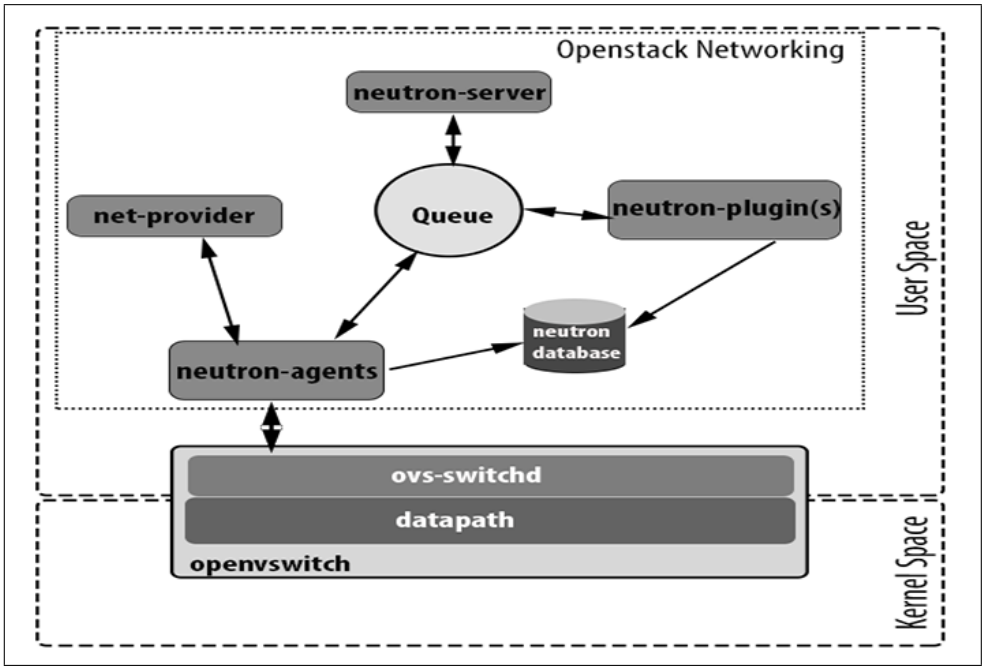

Figure 4.8 Networking Node Architecture

4.7.1 Neutron Analysis

Neutron is an Openstack project which provides networking as a service, thanks to a rich API for buiding networking topologies. It is responsible for virtual networks, subnets and routers life cycle in the infrastructure. We instrumented Neutron in the same way as we did for Nova. However, here we are interested in network and router behaviours. We write the trace events using the JSON format. Table-4.4 displays the available tracepoints that we defined for Neutron diagnosis.

Table 4.4 Neutron events [short list]

sub-services	events
dhcp-agent	network.create.{start/end}
dhcp-agent	network.update.{start/end}
dhcp-agent	network.delete.{start/end}
dhcp-agent	subnet.update.{start/end}
dhcp-agent	subnet.delete.{start/end}
dhcp-agent	port.delete.{start/end}
l3-agent	router.create.{start/end}
l3-agent	router.remove.{start/end}
l3-agent	router.delete.{start/end}

We defined algoritms for Neutron service tasks visualization based on the state provider described in Figure-4.9. We focused our study on workers like the l3-agent, responsible for virtual routers management and dhcp-agent, which handles network and subnet lifecycles. Tracing Neutron is a great way to detect service activities errors and locate the platform weakest links. The views (Figure-4.10) generated retrieve task performance so that we are able to find any bottleneck and understand service interactions.

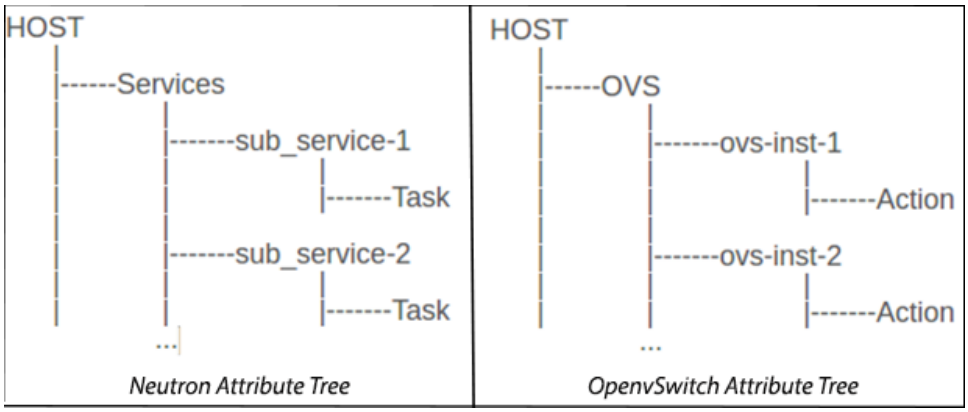

Figure 4.9 Networking Diagnosis Attribute Trees for TraceCompass

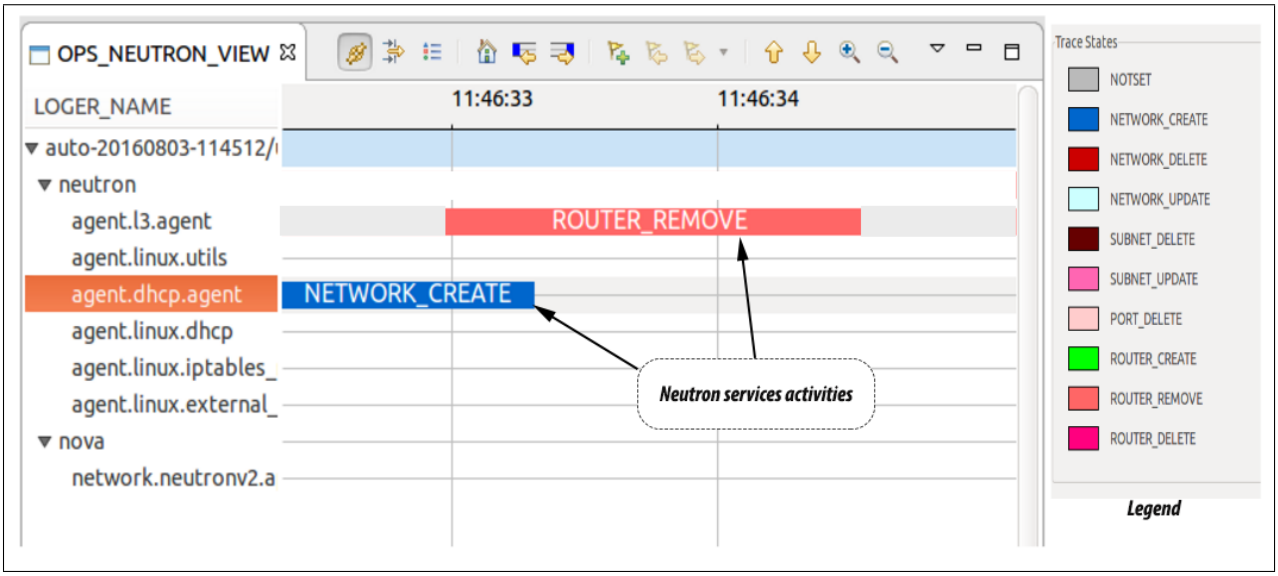

Figure 4.10 Neutron Services Activities

4.7.2 Open vSwitch Analysis

Open vSwitch[Pfaff et al. (2009)] is a multilayer virtual switch acting as an advanced edge switch for virtual machines. It is used to build network topologies. Open vSwitch (OvS) is composed of two main components[Pfaff et al. (2015)] that implement the switch and direct packet forwarding : ovs-vswitchd and kernel datapath. Ovs-vswitchd is a userspace daemon which is the same from one operating system to another. The datapath kernel module is written specifically for the host operating system for fast packet processing.

Figure-4.11 shows how Open vSwitch forwards packets. When the first packet is received by the datapath kernel module from a physical NIC or a virtual machine's VNIC [stage 1], it would be transmited to ovs-vswitchd to get a list of possible actions to perform [stage 2]. At this step, ovs-vswitchd will determine from the 'Flow table' [stage 4] how to handle this packet and will send this information to the datapath. These actions are cached to be used later for similar packets [stage 5,6]. When the datapath gets the result, it applys the actions to the packet and sends it to the specified NIC[stage 7]. For future similar packets, kernel datapath module will directly send them to the destination using actions cached [stage 1 to stage 2]. These packets will be forwarded faster than the first one. The processing path chosen for a packet will impact the OvS ability to forward this packet from one virtual device to another. It is important that Open vSwitch updates the flow table at any network topology modification, to retrieve the best (and updated) actions to perform on the received data.

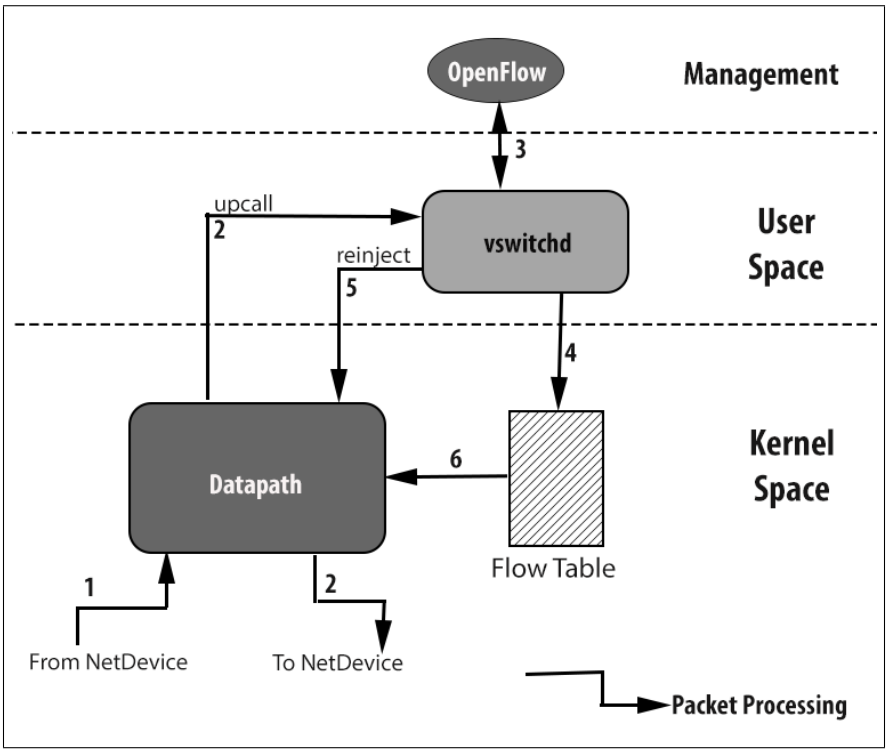

Figure 4.11 Open vSwitch Packet Processing

Mick Tarsel[Tarsel] introduced ovs instrumentation. He showed how to estimate packets upcall events and measure their latency. However, his approach is not applicable for running systems, as the Open vSwitch module should be patched and restarted. An easy way to trace Open vSwitch is to register a probe (using kprobe) with the desired function and record trace events. Giraldeau[Giraldeau] proposed Lttng modules addons for kernel tracing based on probes. In our work, we extend the Tarsel approch and we define new tracepoints that we collect with Lttng-addons. Table-4.5 describes our Open vSwitch tracepoints. To follow a packet through OvS, we implement the same process introduced by Tarsel by tagging its buffer tail room. Each packet received from a network device is tagged with an unique identifier. Then, we are able to detect each stage in which the packet travels.

A packet takes a **slow path** when the datapath module pulls ovs-vswitchd to determine what action to perform. It takes the **fast path** when the information is in cache and the datapath sends it directly to the destinaton NIC. Table-4.5 describes the trace event defined to collect ovs components processing. Each packet received from the NIC will follow the slow path or the fast path (Figure-4.12). An optimal behaviour of our system will be when we have a low frequency of slow paths to forward packets during a communication. To analyse packet path processing with Tracecompass, we define four states which indicate the forwarding stage

Table 4.5 OvS tracepoints

events	description
ovs_vport_receive	ovs receives a packet from a network device
ovs_upcall_start	datapath calls ovs-vswitchd to get a list of actions to perform
ovs_upcall_end	ovs-vswitchd returns possible actions to execute on this packet
ovs_vport_send	datapath sends the packet to the specified network device

(Figure-4.11). Figure-4.12 displays the OvS behaviour : state *in_out* indicates that packets took the fast path. The kernel datapath module directly sends them to the destination vNIC, because it already knew what operation to execute ; state *received* shows that the datapath gets some packets from netdevices, if there is an associated action in its cache, it will be executed as described ; *upcall* reveals that the datapath transmits packets to ovs-vswitchd, waiting for information about the appropriate forwarding operation. *send* when all the actions are performed on a packet it will we be sent to its destination.

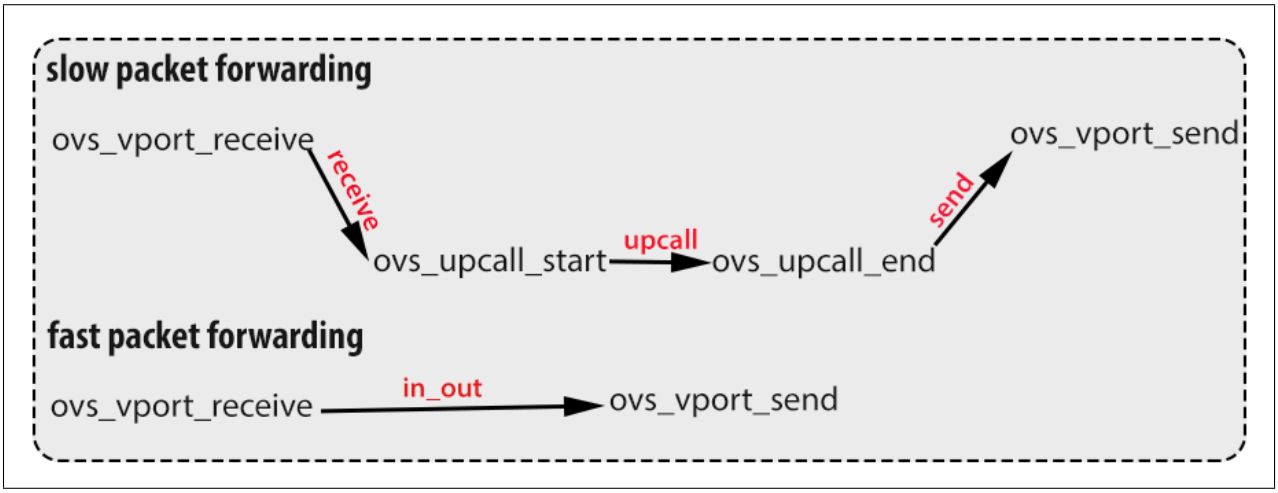

Figure 4.12 Packet forwarding events

Our goal here is to evaluate the OvS performance and retrieve useful data about packets transmission latency. With the collected traces, we are able to display interesting graphical views :

— *Packet switching performance* - We determine the frequency of network topology reconfiguration and its impact during data handling. Figure-4.14 shows the forwarding speed decreasing over the time as OvS components learn how to handle packets, based on their cache information. (Their cache provides a list of actions obtained from upcalls for previous packets). Flow caching helps OvS to consume less hypervisor resources and reduce packet handling time. However, a certain number of tasks can impact the performance of this method and cause a fluctuation of the switching time, as shown by Figure-4.14. Among these tasks, we have *Network reconfiguration*. After a network reconfiguration, the SDN controller will send new OpenFlow tables to ovs-vswitchd, which will match any packets received from the datapath module against these tables and cache the actions in the kernel. This reconfiguration leads packets to take the slow path more frequently. The more the controller updates the flow tables, the more packet handling times increase. Thus, OvS often updates the cache to reduce this latency. Another task is *Flow cache eviction*. Open vSwitch user-space (ovs-vswitchd) gets cache statistics to establish which datapath flow is useful and which should be evicted from the kernel. If packets match the evicted flow, that may lead to redoing upcalls, loosing on packet switching speed.

— *OvS components interaction* - Figure-4.13 describes the state taken by packets through OvS daemons. Each state time defined in TraceCompass will give an idea of the performance of any operation about packet transmission. Our view helps measure packet latency through OvS components and get the time taken by the userspace daemon to get the list of actions to perform on each packet (slow path processing vs fast path).

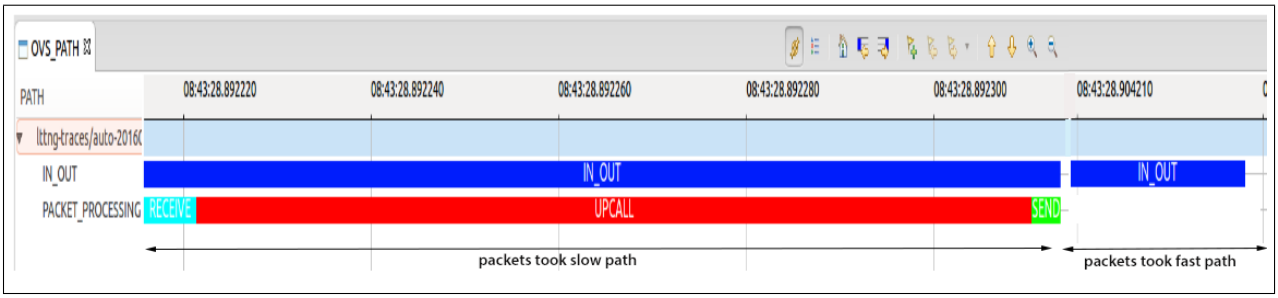

Figure 4.13 OvS VIEW : firsts packets took the slow path but following packets took the fast one

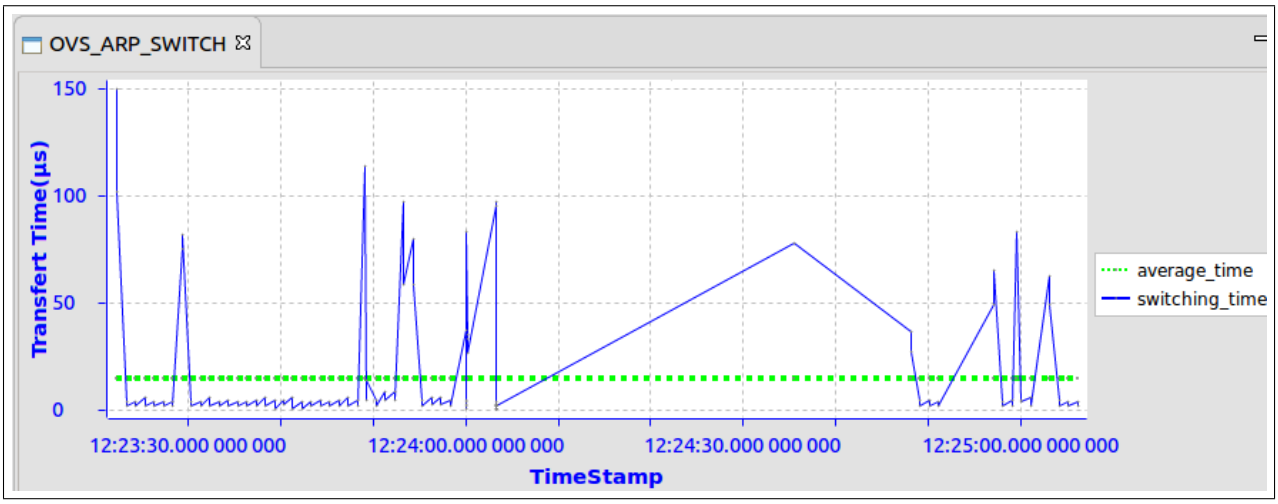

Figure 4.14 ARP Packets Switching

4.7.3 Kernel Analysis

Multilayer tracing of networking services grant enough information to locate abnormal behaviour of the infrastructure. If an issue is due to a controller misconfiguration, we can figure it out with the Neutron trace. Open vSwitch tracing retrieves the host virtual layer problems. However, when latencies are due to some processes interferences (CPU or I/O usage), kernel traces are required to understand them. The kernel analysis for the networking service proceeds as explained in section V. Figure-4.15 presents a case of possible catastrophic slowdown caused by CPU congestion. Indeed, virtual machine processes are using the resource when ovs-vswitchd is waiting its turn. No application (deployed in the guest) which is waiting for data through the network, will receive data until the guest reduces its CPU usage to let OvS perform packets switching.

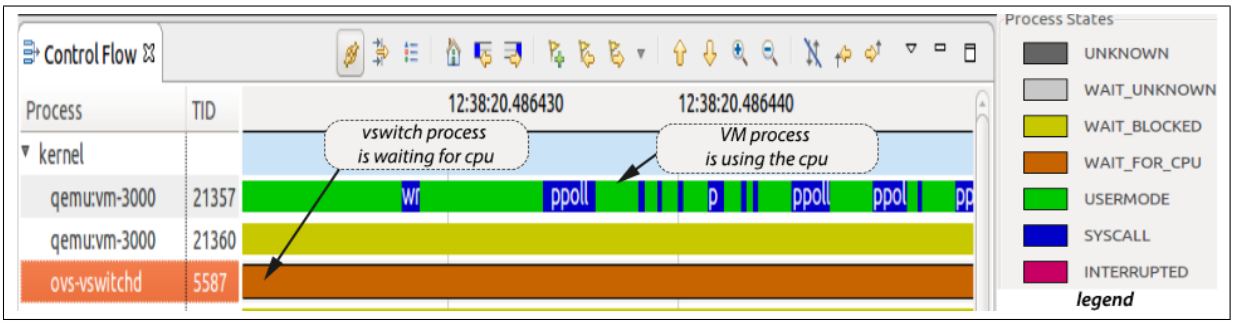

Figure 4.15 OvS Preemption

4.8 Storage Service Diagnosis

In this paper, we analyse cloud storage according to its usage from virtual machines. Nonetheless, it is also possible to diagnose it as an independent service. We want to verify that the manager does not attach the same disk volume to different virtual machines, causing some contention issues. Moreover, the storage service investigation will help verify that there is no performance issues about disk I/O tasks, and ensure that disk space is released correctly after usage. In our work, we deployed OpenStack Cinder to provide storage services.

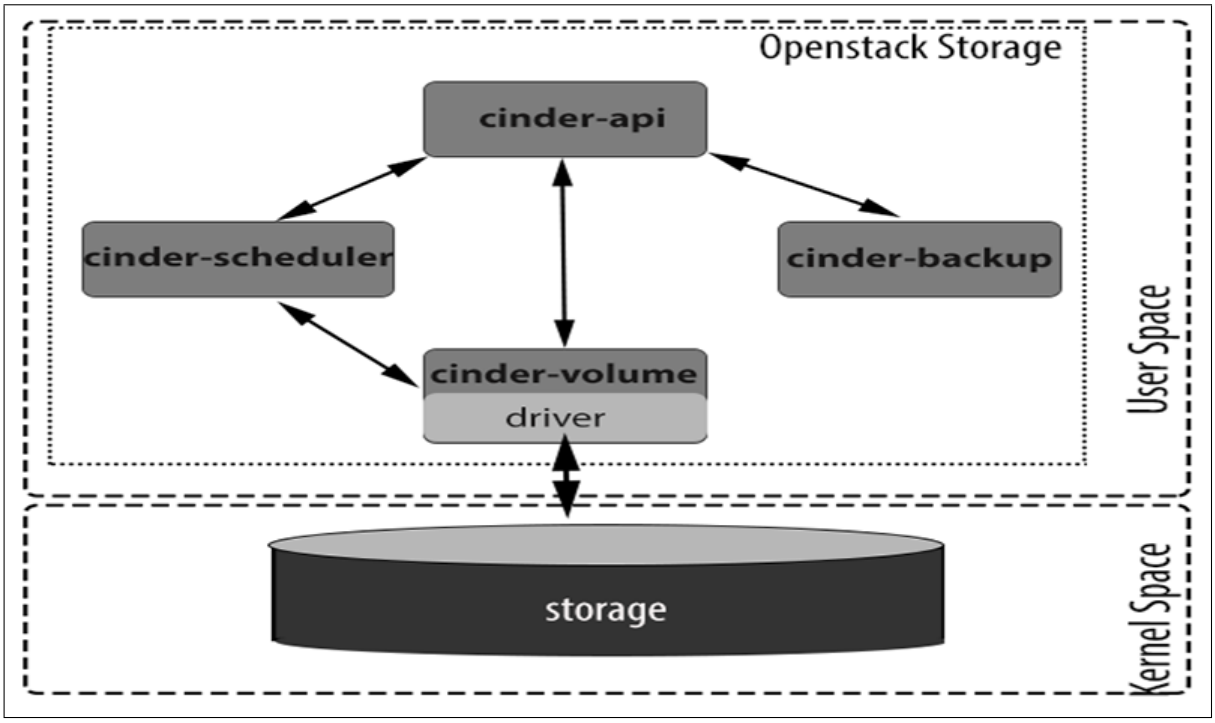

Figure 4.16 Storage Node Architecture

4.8.1 Cinder Analysis

Cinder is an Openstack storage service project designed to provide storage resources to consumers. It virtualizes storage devices and provides some API for end user requests. Thanks to the Cinder interface, the consumer does not have to worry about where the current volume is really located in the cloud system. A most common usage scenario for Cinder is to provide persistent storage to virtual machines. The Cinder project provides a lot of useful logs for troubleshooting. Nevertheless, we chose to rewrite this in a more structured way to provide data in the JSON format, like for the Nova and Neutron projects. We are retrieving 'volume instances' data and the related VMs using them. Table-4.6 presents some Cinder tracepoints,

more details are available online at [Bationo (c)]. For this work, the analysis view generated

Table 4.6 Available tracepoints for Cinder

Resource type	events
volume	volume.create.{start/end}
volume	volume.update.{start/end}
volume	volume.attach.{start/end}
volume	volume.detach.{start/end}
volume	volume.migrate.{start/end}
snapshot	snapshot.create.{start/end}
snapshot	snapshot.delete.{start/end}

with TraceCompass is based on the attribute tree shown in Figure-4.17, which relies on volume state and service actions. Cinder tracing provides useful data for developers and system administrators alike. Our framework displays trace events for debugging. We also compute the performance of Cinder operations by measuring the time taken to complete any volume life-cycle task, like a volume creation or its attachment to a VM, as presented in Figure-4.18. In addition we are able to link any virtual volume to a physical disk, and pinpoint the VM which is using it.

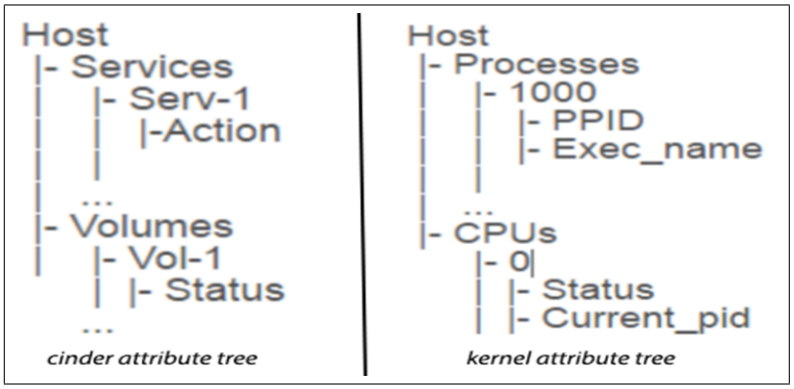

Figure 4.17 Storage Diagnosis Attribute Trees for TraceCompass

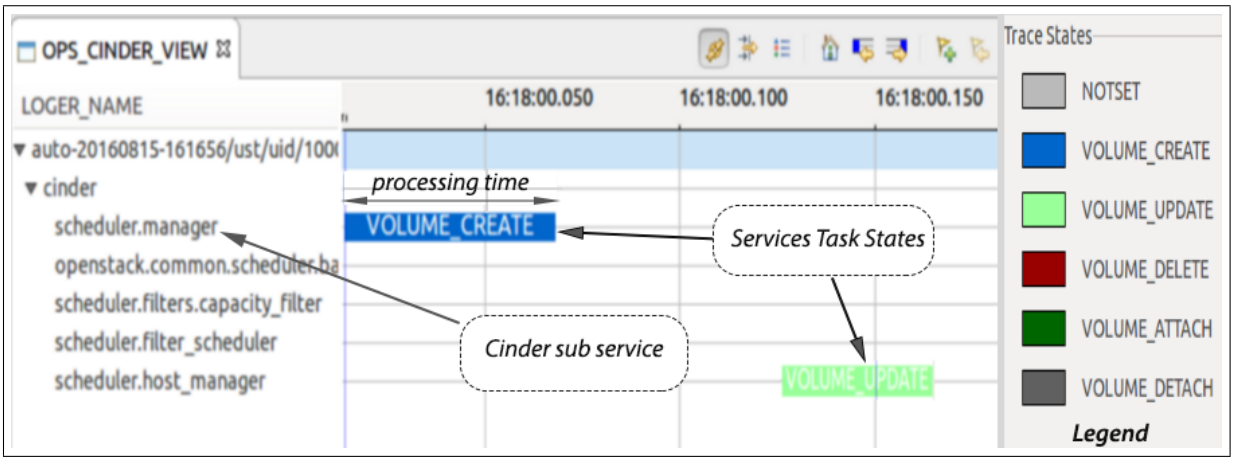

Figure 4.18 Cinder Services View

4.8.2 Kernel I/O Analysis

Virtual machine activities can suffer from physical system operation issues like computing, networking or storage disk I/O latency. When the performance degradation is due to Disk usage, it is important to have an efficient tool to display the problem. Fortunately, TraceCompass incorporates a Disk I/O view to check in details storage tasks. Nonetheless, virtual machine tasks troubleshooting requires more knowledge about their current volume location. In this paper, we provide those links through the Cinder Component tracing. Then, any VM is associated to its poll of virtual disks, which are located in some known physical disks. Kernel tracing produces enough information to get disk read and write processing time (Figure-4.19) and find disk usage contention.

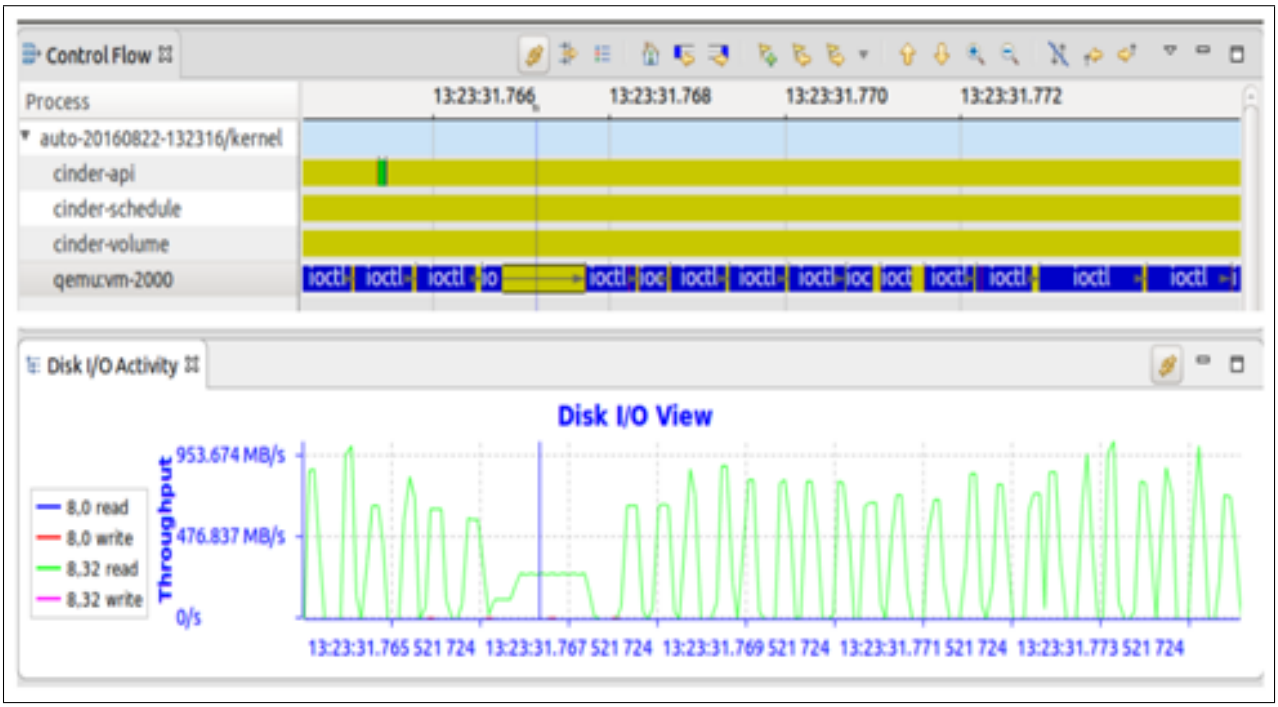

Figure 4.19 Disk IO Activity

4.9 Experiments : Live migration

In this section, we apply multi-layer tracing to one of the most difficult problems in cloud monitoring : live migration. Indeed, in that case, more hosts are involved and some operations start on one host and continue on another, which significantly complexifies the tracing operation. We thus performed some experiments about guests migration. We recorded a trace of our cloud installation and achieved some analysis. The traces used in this paper are available online[Bationo (c)]. Here, we focus on virtual machine migration analysis as a cloud computing services operation only, networking and storage issues are not part of the scope here. Live migration is a technique to move a virtual machine from one physical host to another. Among the virtualization features, it is one of the most important, providing resiliency and resource availability in the cloud environment. This operation is used in many cases to reach performance objectives. For example, when the guest workload increases, a migration operation will relocate it in a new host, with more resources to satisfy its demands. Live migration analysis requires the gathering and correlation of the information from the three layers, both on the source, the destination and the controller hosts. On the Nova side, migration is performed as describe by Figure-4.20.

i. *select destination host* (controller) : find compatible destination hosts in the cloud environment. These physical machines must support the virtual machine parameters and pass the filtering step. The scheduling service will pick up one of them as a destination host.

ii. *start migration process* (source host) : start migration on the source host, tell the destination to be ready for the operation.

iii. *pre-migration step* (destination host) : get the block device and network information of the instance and prepare to setup the network on the destination.

iv. *migration monitoring* (source host) : contact QEMU to proceed with the migration, poll the QEMU monitor to get the operation state until it finishes or fails. Rollback the process if the migration time exceeds the default duration specified, or if migration fails.

v. *post-migration* (source host) : detach the volume connection from the instance, disable the network address, unplug the virtual network interface on the source host, clean up and update the Nova database.

vi. *post-dest-migration* (destination host) : setup a network on the destination host and update the instance information ; the virtual machine is fully running on the destination.

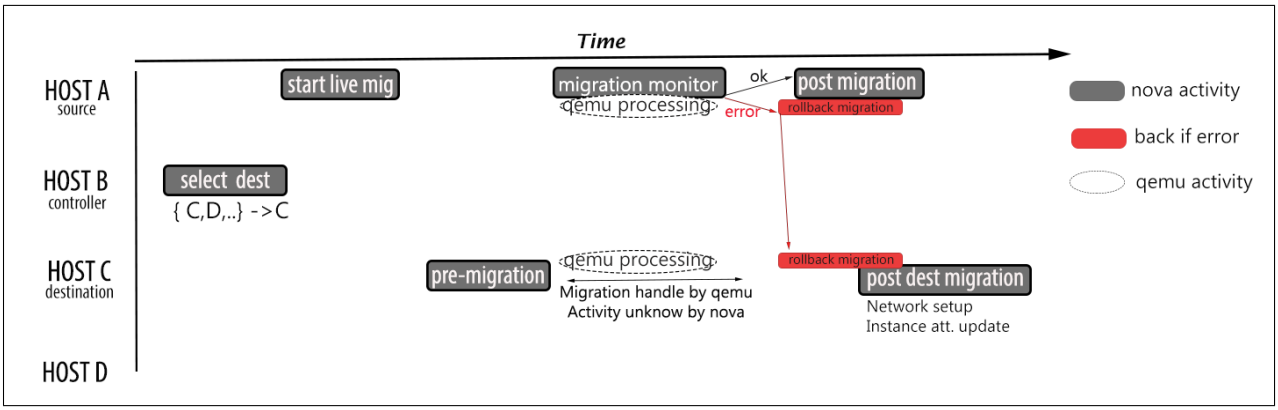

Figure 4.20 Nova Live Migration activity

The Nova-compute service polls the hypervisor to get the current state of the migration process. As migration is handled by QEMU, Nova does not have access to the migration internal state. Therefore, when something goes wrong, Nova cannot find what is happening but will perform a rollback task by releasing the resource and cleaning up the destination machine.

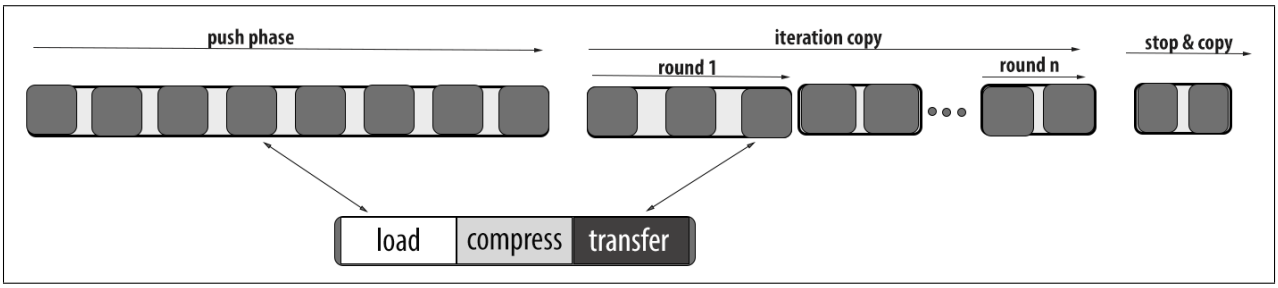

Figure 4.21 Live Migration : QEMU steps

We complete these high-level informations with QEMU events, for more details about the operation. QEMU proceeds with the migration in three steps (Figure-4.21) : during the first step, all the memory is moved to the destination. In the iterative copy phase, the dirty memory is transferred in several rounds until there is not enough dirty-data. Next, in the stop-and-copy phase, the virtual machine is stopped at the source, the memory is moved, and the VM is resumed at the destination. To improve the performance analysis, we defined some sub-steps for each [Figure-4.21] : the **load** stage when the memory is loaded by QEMU, the **compress** stage describes the compression of the memory, the **transfer** step is when the memory content is sent to the destination.

Live migration has to deal with a lot of challenges that can reduce the QoS of applications running inside the virtual machine. Moreover, increasing the number of guests in a host leads to VMs concurrently competing for physical resources utilisation, which causes service degradation. Bloch and al.[Bloch et al. (2014)] identify some interferences in the cloud environment which possibly impact negatively upon a live migration process.

— Co-location interference - VM migration requires extra CPU usage and the bandwidth to move guest data. Unfortunately, the productivity of applications running inside the physical host will be reduced.
— Network interference - network I/O virtualization suffers from virtual interruptions to guests VMs. Moreover migration bandwidth usage degrades communication among VMs.
— Resources interference - migration reduces resource availability and raises CPU or memory contention. At the destination, a memory space will be assigned for the newly created VM and that operation will disturb the existing VMs. However, live migration will take more time to complete if there is not enough spare CPU.

In this paper, we performed three experiments on virtual machine live migrations and report about the results. There are several parameters used to evaluate live migration like

the *Migration time*, which refers to the total time used to copy the device memory from one physical host to another and resume the migrated VM. The *downtime* is the time interval during which the VM is not running during the stop-and-copy step. Clark [Clark et al. (2005)] demonstrates that the downtime is critical, as it may cause service disruption. In the literature, the migration downtime is estimated by polling the virtual machine interface until it responds, but that is a relatively coarse measure. Here, we get the downtime by deducing it from the QEMU events. The downtime is related to the stop and copy time in our QEMU activity view(Figure-4.22). We are able to estimate with high accuracy the memory copy stages, and the downtime, using the algorithm displayed below. For these live migration experiments, we proceeded according to the following steps : (a)-start tracing OpenStack Nova, QEMU, and kernel of the controller, source and destination hosts using LTTng streams to collect trace files automatically on a specified node. (b)-run the live migration process. (c)-stop tracing after the migration completes. (d)-upload the trace in TraceCompass to synchronise and generate the migration view.

Algorithm 1 GETTING MIGRATION STAGE

```
events ← getEventsOfVM(vm-name)
map_col ← getBitMapEvents(events)
Precopy ← map_col.get(0).time - map_col.get(1).time
i ←0
while map_col.hasNext() do
    start ← map_col.prev()
    end ← map_col.next()
    Iteration_round_(i) ← end.time() - start.time()
    i ← i + 1
start ← map_col.lastItem()
end ←  getEvent('kvm_run')
Stop&Copy ← end.time - start.time
```

4.9.1 Experiment-1

We proceeded with the first test by migrating a virtual machine and retrieving all possible traces from three layers at each node. We reduced the environment's activity to be sure that no process will interfere with the migration task. Table-4.7 shows the result we get from the high-level layer about the main parameter. As expected, the downtime is unknown at this stage, that is why we need the QEMU level trace. The performance of the first experiment serves as a benchmark when studying other experiments. Figure-4.22 describes all the stages of the migration, the number of copy-rounds and the downtime estimation.

Table 4.7 Experiments performance results from high-level

Operation	experiment1	experiment2	experiment3
$SELECT_{dest}$	11.22ms	13.611ms	31.69ms
$PRE\text{-}MIGR_{time}$	494.60ms	524.23ms	234.33ms
$POST\text{-}MIGR_{time}$	2.55ms	-	3.56ms
$POSTDESTMIGR_{time}$	111.19ms	-	372.47ms
$TOTALMIGR_{time}$	166.12sec	-	169.320sec
DOWNTIME	unknow	-	unknow
$MIGR\text{-}END_{state}$	ok	failed	ok

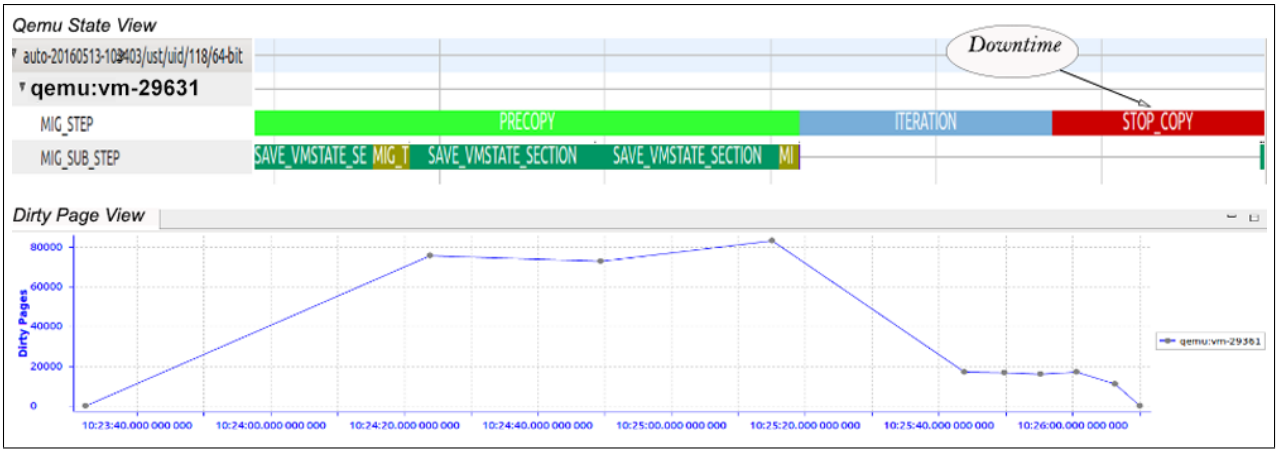

Figure 4.22 Experiment-1 : Migration succeeds ; dirty-pages curve converges

4.9.2 Experiment-2

For the second experiment we tried to migrate a virtual machine with a high workload : we stressed the VM with a stress tool to increase the CPU and memory usage.
In this case, the VM never moved to the destination host so the operation failed. The OpenStack service view on Figure-4.23 shows a rollback process after the migration starts and, according to Figure-4.20, a rollback only happens if we have some issues like a bug in the virtualization layer, the migration timeout is reached, or network contention interrupts the data copy. By analysing the middle layer using the QEMU state view, we found only two steps in the migration operation : precopy and iteration copy. The stop and copy stage is missing, confirming that the migration has never completed. More details are provided by the QEMU dirty page view by returning a curve which does not converge to 0, unlike in the first experiment. In this case, we conclude that the migration failed due to the virtual machine workload ; the page dirtying rate is higher than the data copy speed to the destination host.

QEMU cannot transfer quickly enough the memory, the task finally reached the migration timeout, and OpenStack launched the rollback process. In fact, during the migration iterative stage, all the dirty memory is copied to the destination until we reach a minimal amount of memory. If the VM continues to dirty the pages already sent, the migration takes more time to complete. To help the migration process go faster, QEMU will slow down the VM by throttling its VPCUs to reduce its execution time. This throttling mechanism rate is increased until the migration process is completed. However, this mechanism was insufficient in that case and the only way to move this guest to the destination is to stop it at the source and resume it at the destination, completely stopping its activity during that interval.

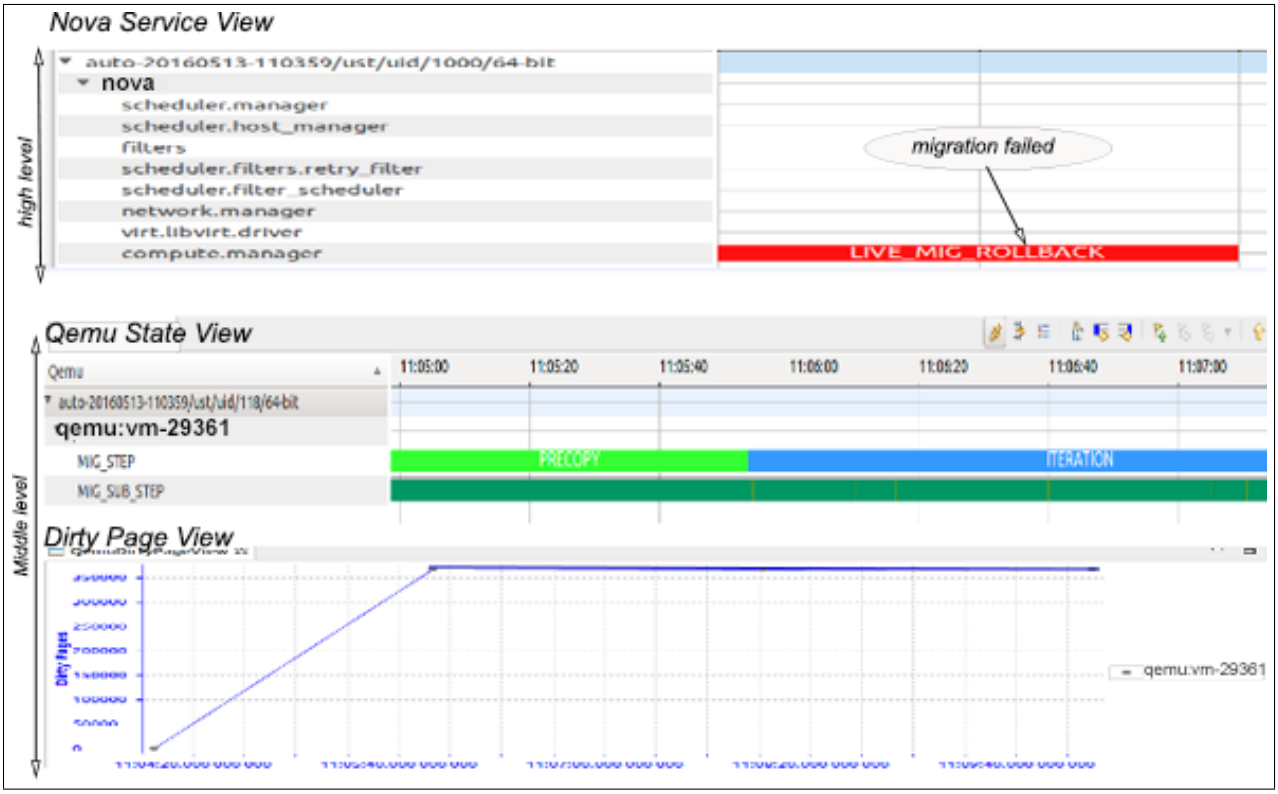

Figure 4.23 Experiment-2 : Migration fails ; dirty-pages curve does not converge

4.9.3 Experiment-3

Here, we analyse the impact of VM interferences during live migration. We try to migrate a VM co-located with four virtual machines with high resource usage. To increase the interference, we use a stress tool on these guests. The migration succeeds but takes much more time than in the first experiment (Table-4.7). This behaviour is explained by the low-level

layer analysis with the kernel trace. In this case, the VMs share the physical resources with other processes of the host. For illustration, in Figure-4.24, the virtual machine vm-23405 is waiting for CPU while other VMs are using the resources. Any process with high priority will preempt the CPU and reduce the vm-23405 access time. Moreover, the preemption of the migrated virtual machine increases the migration time. The interference of co-located VMs really matters when we want to perform migration quickly. As a VM is preempted more often, its activities take more time to complete. We can estimate the virtual machine preemption rate, and thus explain its service latency and deduce that it is better to increase, if possible, the VM process priority before moving it to another host.

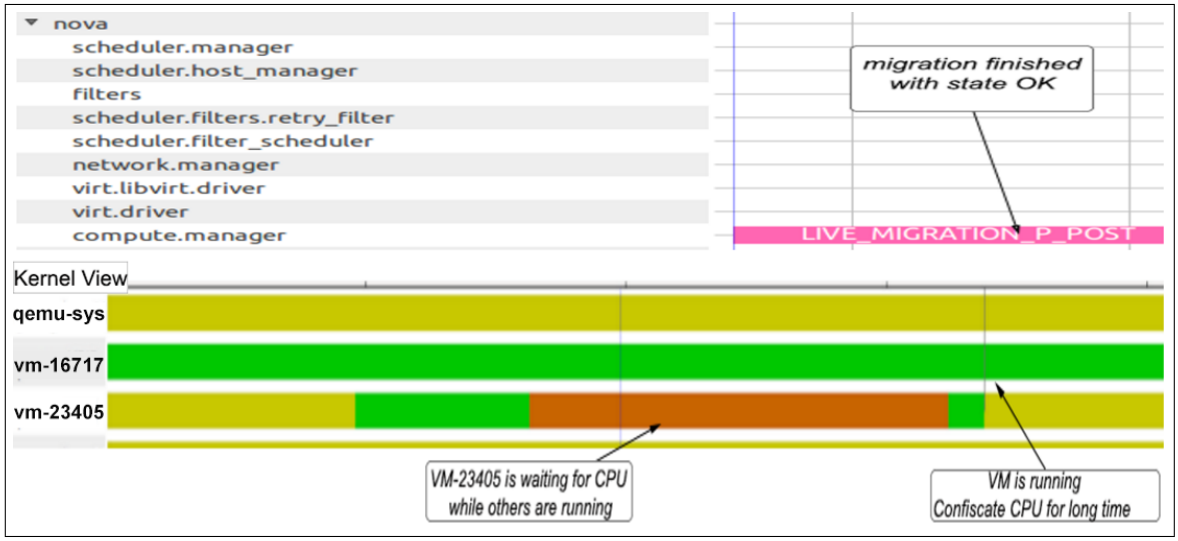

Figure 4.24 Experiment-3 : vm-2340 migration takes much more time to complete

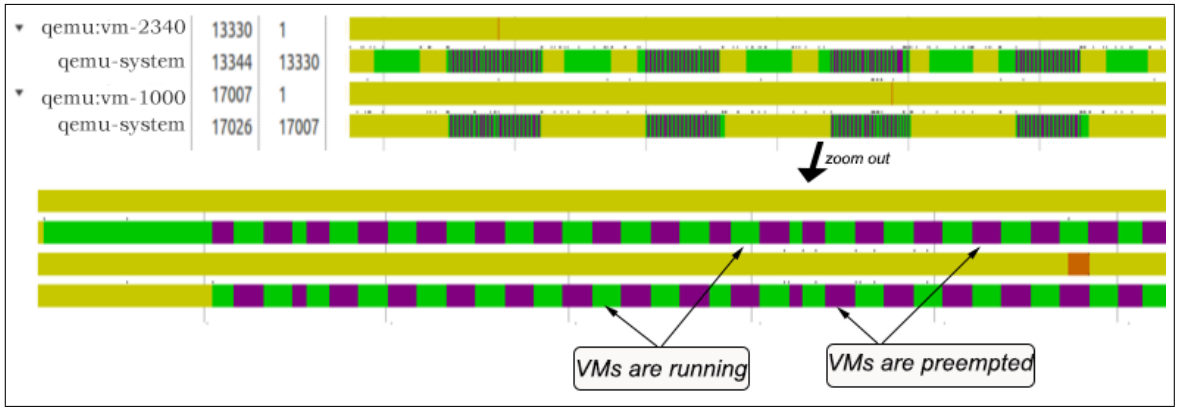

Figure 4.25 VM Preemption : vm-2340 interferes with vm-1000

4.10 Conclusion

Cloud services troubleshooting is a real challenge nowadays because it requires gathering and correlating data, from the infrastructure hosts, at several layers in many nodes. In this paper, we introduced cloud services multi-layer tracing, and we demonstrated its effectiveness by collecting data, from all the cloud nodes layers, and detecting and diagnosing service issues like virtual machine live migrations failures or performance degradations. We also presented analysis and views for the other main cloud services like Nova (computing), Neutron (networking) and Cinder (storage services). Indeed, we define the needed identifiers required to associate events from traces at the different layers, to follow user requests from the application level to the kernel layer, and diagnose more easily performance problems.
As future work, we will evaluate the networking services performance with a platform analysis of Network Function Virtualization. Furthermore, we will search for guidelines about cloud service threats through kernel layer tracing. We plan to detect security issues through virtual machines tracing in a cloud computing infrastructure.

4.11 Acknowledgment

The authors would like to thank NSERC, Prompt, Ericsson and EfficiOS for funding this research project. They also thank Francis giraldeau for help in the Lttng-Addons modules, and Mohamad Gebai and Hani Nemati for reviewing this paper.

CHAPITRE 5 DISCUSSION GÉNÉRALE

5.1 Retour sur l'analyse multicouche

5.1.1 Modèles de diagnostic

La surveillance des infrastructures se fait selon deux types d'analyse. Ces deux types peuvent être combinés ou utilisés indépendamment pour comprendre les états des machines physiques et virtuelles.

Analyse horizontale de la plateforme

Ce qui nous intéresse, c'est l'interaction entre des éléments qui appartiennent à la même couche, mais qui sont situés sur des machines différentes. L'analyse va consister à collecter les traces de cette couche sur chaque machine, pour suivre les temps de requête et de réponse entre ces éléments. Par exemple, si nous cherchons à déterminer le temps de communication entre le composant nova-scheduler [1] de l'hôte contrôleur et le composant nova-compute [2] de l'hôte compute1, il faudrait agréger et synchroniser les traces Nova de ces deux machines dans Trace Compass.

En rappel, la synchronisation requière l'activation des points de traces *inet_sock_in* et *inet_sock_out* sur les machines impliquées dans l'analyse.

Analyse verticale de la plateforme

Dans ce cas, notre diagnostic est focalisé sur les couches d'une même machine. Notre intérêt est de connaitre les temps d'exécution entre les instances des différentes couches du système. Par exemple, nous cherchons à déterminer la réactivité de QEMU face aux requêtes de nova-compute. Ce type d'analyse ne requiert pas de synchronisation de traces. En plus, elle est moins coûteuse en bande passante que l'analyse horizontale.

5.1.2 Contraintes de l'analyse multicouche

Le diagnostic multicouche introduit quelques contraintes de configuration et de déploiement des services. Il nécessite de retrouver les instances des ressources virtuelles à tous les niveaux et d'associer leurs informations. Cependant, la nomenclature des ressources virtuelles, utilisée par défaut par OpenStack et QEMU, ne permet pas de les identifier de manière précise. Aussi, pour des raisons de cohérences des données, il est nécessaire d'appliquer les paramètres exposés à la section [3.4.3] pour changer la nomenclature.

1. nova-scheduler est responsable du choix de l'hôte de chaque VM
2. nova-compute est le composant du service Nova responsable du cycle de vie des VMs

Nous avons diagnostiqué séparément les modèles de services (calcul, réseau, stockage). Une analyse globale des services pourrait provoquer des perturbations et fausser le diagnostic du comportement des systèmes. En effet, la surveillance de chaque service nécessite l'activation de différents points de trace qui induisent plus de latence : plus nous aurons des évènements, plus nous provoquerons des interférences.

5.2 Surcoût du traçage

Nous avons traité l'analyse des performances d'exécution des systèmes à l'aide des traces multicouches. Toutefois, en milieu de production, les services infonuagiques peuvent être perturbés par les outils de traçage. Pour trouver la latence induite par le processus de traçage sur une opération donnée, nous calculons la différence entre le temps mis pour compléter cette opération lorsque le traçage est activé et le temps mis pour terminer l'opération quand le traçage est désactivé. Plusieurs travaux[Giraldeau and Dagenais (2016)], sur les coûts engendrés par le traçage, ont démontré que LTTng induisait une latence plus ou moins égale à *200ns* par évènement (cela est inférieur à celles des autres outils). Dans notre cas, nous pouvons envisager des ralentissements à trois niveaux de l'hôte diagnostiqué.

Couche application

Les points de traces qui sont ajoutées dans OpenStack (Nova, Neutron, Cinder) pourraient causer des latences. Cependant, les tâches de cette couche génèrent peu d'évènements (en moyenne deux - une pour le début et une pour la fin de la tâche). En plus, les mécanismes de communication des ressources virtuelles avec l'hôte ne sont pas gérés par la couche application : cela rend le traçage peu onéreux. Par exemple, l'expérience *Experiment-1*[4.9.1], exposée dans le chapitre[4], a donné un taux de 3 évènements par seconde, soit une latence d'environ $100\mu s$ sur les *166s* du temps total.

Couche virtualisation

C'est une couche sensible qui ne doit pas être tracée de manière inconsidérée pour ne pas provoquer de grande latence. Dans le but de limiter les baisses de performance, nous n'avons activé que 5 points de trace pour investiguer la migration à chaud. Le traçage a provoqué une latence plus ou moins égale à *1.6ms* au cours de *Experiment-1*[4.9.1].

Couche noyau

Plusieurs processus font appel à des fonctions systèmes et cela rend l'estimation de la quantité d'évènements moins aisée. Mais on peut avoir une idée en dénombrant les évènements ayant le même pid que la VM migrée. Dans le cas de la migration de *Experiment-1* [4.9.1], la baisse provoquée tournerait autour de *1s* sur les *166s* qu'a duré l'opération.

Nous estimons la latence causée par l'analyse multicouche à environ 1% du temps de migration quand le traçage est désactivé.

CHAPITRE 6 CONCLUSION ET RECOMMANDATIONS

Dans ce chapitre, nous revenons sur les travaux effectués, et nous présentons les limites de la solution proposée. Et pour finir, nous proposons des idées de travaux futurs qui pourront améliorer cette solution.

6.1 Synthèse des travaux

6.1.1 Atteinte des objectifs

Nous nous sommes proposé d'examiner les anomalies dans les infrastructures de type IaaS (Infrastructure as a Service) [1.1.2] qui utilisent OpenStack comme application de gestion de services. Le but de notre recherche était de concevoir un outil permettant de diagnostiquer les erreurs liées à l'exécution des environnements virtuels. Ainsi, nous répondions à la question : *est-il possible, par agrégation des traces de l'espace noyau et des traces de l'espace utilisateur, d'investiguer les sources de baisse de performance et de détecter les troubles de fonctionnement des services infonuagiques* ?
Nous savons que les services d'OpenStack communiquent entre eux à travers un canal de messagerie comme RabbitMQ. Il a été possible, au début de notre recherche, de retrouver les messages et les notifications envoyés par ceux-ci, en écoutant les fréquences de transmission dans RabbitMQ. Toutefois, les résultats se sont avérés insuffisants pour comprendre les opérations dans la plateforme. Nous décrivons, en Annexe [A], le mécanisme adopté pour cette analyse. Ensuite, nous avons opté pour une seconde méthode qui consiste à modifier le code d'OpenStack, pour trouver des données plus détaillées sur les tâches exécutées, en ajoutant des points de trace. Cette méthode constitue l'essentiel de notre diagnostic sur la couche application.

Au vu des résultats obtenus au cours de nos expériences, nous pouvons confirmer que nous avons atteint les objectifs de notre étude. Cet ouvrage expose les résultats de nos travaux. Dans l'introduction, nous avons établi la problématique et introduit les modèles de services infonuagiques. Nous avons présenté une revue de littérature concernant les outils et les notions nécessaires pour comprendre la surveillance et la mesure des performances des systèmes. Cette revue de littérature nous a permis de prendre connaissance des outils de traçage et des technologies de virtualisation existantes. Puis, nous avons expliqué, dans le chapitre [3], la méthodologie appliquée dans notre travail. L'environnement de recherche, les configurations et les étapes d'investigation de cette étude y ont été expliqués. Ensuite,

l'essentiel de la solution est présenté au chapitre [4]. Dans le dernier chapitre, nous abordons une critique de notre solution et les surcoûts liés au traçage multicouche.

6.1.2 Contributions pertinentes

Comme contribution, nous avons proposé des solutions pour une investigation des performances et pour la détection des anomalies dans les infrastructures infonuagiques.

- D'abord, nous avons établi un modèle d'instrumentation d'OpenStack qui est considéré comme étant la couche d'application. Nous avons instrumenté les projets Nova, Neutron, et Cinder. Le même principe peut être appliqué aux autres projets d'OpenStack.
- Ensuite, pour faire ressortir les évènements liés au traitement et au routage de paquets, nous avons proposé une instrumentation d'Open vSwitch.
- Puis, nous avons fourni des interfaces graphiques pour analyser les traces des couches applications, en utilisant Trace Compass. Ainsi, nous pouvions retrouver toutes les opérations et les interactions des services de Nova, de Neutron et de Cinder.
- Après, une analyse des traces des couches de virtualisation a été proposée. Nous avons pu investiguer les états de QEMU et les performances de Open vSwitch.
- Enfin, dans l'article présenté au chapitre [4], nous avons proposé une analyse multicouche des opérations de migration à chaud.

6.2 Limitations de la solution proposée

L'investigation des performances de OvS est basée sur le marquage des paquets qui transitent à travers lui. Elle ne donne pas toujours les résultats réels, car le mécanisme de marquage qui consiste à identifier les paquets, en ajoutant de l'information dans sa queue (tailroom), n'est pas assez efficace. Certains paquets arrivent avec des queues déjà pleines, ce qui rend impossible l'ajout supplémentaire de données. Ces paquets seront ignorés dans notre analyse, car il est difficile de suivre leur parcours. Puisque LTTng capture tous les évènements en rapport avec les traitements des paquets, nous pouvons calculer la quantité de paquets qui vont être ignoré, en se basant sur les évènements de type ovs_vport_receive[3.3] avec des paquets d'identifiant 0.
De nos différentes expériences, nous estimons, par le rapport [*nombre d'évènements vport_receive avec paquet_id=0/nombre d'évènements total vport_receive*], que la quantité de paquets ignorés est d'environ 10% à 15% des paquets transitant dans OvS. Toutefois, cette limite n'entache pas la crédibilité des résultats, car nous obtenons une estimation moyenne qui est suffisante, pour dire si notre service de réseau est assez performant ou pas.

L'analyse de la plateforme requiert de collecter les données des machines vers un système réservé pour la visualisation des traces en utilisant *LTTng-Streaming*. La collecte peut engendrer une consommation excessive de la bande passante et perturber les tâches des services infonuagiques. Ainsi, une investigation en ligne [1] est à proscrire dans un environnement de production où les services sont très dépendants de la bande passante. L'idéal serait de procéder à un diagnostic hors ligne [2] (en différé) de la plateforme. Cela consisterait à tracer les machines localement et à rassembler les traces bien plus tard pour l'analyse : le traçage local est moins onéreux que le streaming, pour ce qui est de la perturbation du réseau. Néanmoins, l'utilisation du streaming reste envisageable, si l'on réduit le champ d'investigation aux machines et aux services que l'on soupçonne d'être moins performants.

6.3 Améliorations futures

Pour rendre plus efficace le diagnostic des services infonuagiques, certaines améliorations peuvent être apportées à notre solution. Tout d'abord, l'adoption d'un format standard pour la représentation des logs dans OpenStack, comme le format JSON utilisé ici, pourrait faciliter le diagnostic des logs déjà présents dans la version officielle d'OpenStack.
Ensuite, il faut noter que les diagnostics des services de stockage et de réseau, qui utilisent respectivement Cinder et Neutron, fournissent peu d'information concernant les machines virtuelles auxquelles ils sont souvent associés. L'analyse deviendrait plus efficace, si nous combinons leurs traces avec celle de Nova pour mieux identifier les ressources virtuelles. Les vues de Nova, de Neutron et de Cinder seraient alors fusionnées pour retourner plus de détails.

Nous avons proposé, dans cette étude, un moyen, pour mesurer les performances de OvS. Cependant, il n'est pas encore possible de situer les anomalies sur les réseaux virtuels. Comme amélioration future, on pourrait tenir compte des adresses sources et destination des paquets analysés, pour retrouver les réseaux et les machines virtuelles qui subissent les pertes de performance.

Enfin, pour une meilleure investigation des systèmes, nous suggérons de fusionner les vues de Trace Compass qui décrivent l'état des services en espace utilisateur et en espace noyau, et fournir des liens entre les opérations des différentes couches de la plateforme. Pour chaque tâche de la couche application, nous pourrions faire ressortir, de manière plus détaillée, les opérations des couches sous-jacentes.

1. online trace analysis
2. offline trace analysis

RÉFÉRENCES

(2015) Tracecompass. En ligne : https://projects.eclipse.org/projects/tools/tracecompass

(2014) Virtual networking. En ligne : http://wiki.libvirt.org/page/VirtualNetworking#Virtual_Networking

R. Acuña, Z. Lacroix, et R. A. Bazzi, "Instrumentation and trace analysis for ad-hoc python workflows in cloud environments", dans *2015 IEEE 8th CCC*.

A. Anand, M. Dhingra, J. Lakshmi, et S. K. Nandy, "Resource usage monitoring for kvm based virtual machines", dans *Advanced Computing and Communications (ADCOM), 2012 18th Annual International Conference on*, Dec 2012, pp. 66–70. DOI : 10.1109/ADCOM.2012.6563586

S. Bagnasco, D. Berzano, A. Guarise, S. Lusso, M. Masera, et S. Vallero, "Monitoring of iaas and scientific applications on the cloud using the elasticsearch ecosystem", dans *Journal of Physics : Conference Series*, vol. 608, no. 1. IOP Publishing, 2015, p. 012016.

P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, et A. Warfield, "Xen and the art of virtualization", *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003. DOI : 10.1145/1165389.945462. En ligne : http://doi.acm.org/10.1145/1165389.945462

Y. Bationo. nova trace analysis. En ligne : https://github.com/yvesjunior/openstack-service-analysis

——. Open vswitch tracing library. En ligne : https://github.com/yvesjunior/ovs-tracing

——. Openstack tracing library. En ligne : https://github.com/yvesjunior/Openstack_tracinglibrary

——. Rabbitmq trace analysis. En ligne : https://github.com/yvesjunior/openstack-service-analysis/tree/master/rabbitmq

F. Bellard, "Qemu, a fast and portable dynamic translator." dans *USENIX Annual Technical Conference, FREENIX Track*, 2005, pp. 41–46.

M. T. Bloch, R. Sridaran, et C. Prashanth, “Analysis and survey of issues in live virtual machine migration interferences.” *International Journal of Advanced Networking & Applications*, 2014.

R. Buyya, J. Broberg, et A. M. Goscinski, *Cloud computing : Principles and paradigms*. John Wiley & Sons, 2010, vol. 87.

D. H. Chau, C. Faloutsos, H. Tong, J. I. Hong, B. Gallagher, et T. Eliassi-Rad, “Graphite : A visual query system for large graphs”, dans *2008 IEEE International Conference on Data Mining Workshops*, Dec 2008, pp. 963–966. DOI : 10.1109/ICDMW.2008.99

C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, et A. Warfield, “Live migration of virtual machines”, dans *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, série NSDI’05. Berkeley, CA, USA : USENIX Association, 2005, pp. 273–286. En ligne : http://dl.acm.org/citation.cfm?id=1251203.1251223

J. Danjou, “open stack cloud software : Ceilometer, the openstack metering project”, *available at julien. danjou. info/blog/2012/openstack-metering-ceilometer*, 2012.

J. Denton, *Learning OpenStack Networking (Neutron)*. Packt Publishing Ltd, 2015.

A. Desai, R. Oza, P. Sharma, et B. Patel, “Hypervisor : A survey on concepts and taxonomy”, *International Journal of Innovative Technology and Exploring Engineering*, vol. 2, no. 3, pp. 222–225, 2013.

M. Desnoyers. Common trace format (ctf) specification (v1.8.2). En ligne : http://git.efficios.com/?p=ctf.git;a=blob_plain;f=common-trace-format-specification.md

——, “Low-impact operating system tracing”, Thèse de doctorat, Ecole Polytechnique de Montreal, 2009.

M. Gebai, F. Giraldeau, et M. R. Dagenais, “Fine-grained preemption analysis for latency investigation across virtual machines”, *Journal of Cloud Computing*, vol. 3, no. 1, p. 1, 2014.

F. Giraldeau et M. Dagenais, “Wait analysis of distributed systems using kernel tracing”, *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2016. DOI : 10.1109/TPDS.2015.2488629

F. Giraldeau. Lttng-addons. En ligne : https://github.com/giraldeau/lttng-modules/tree/addons/

F. Giraldeau, J. Desfossez, D. Goulet, M. Dagenais, et M. Desnoyers, “Recovering system metrics from kernel trace”, dans *Linux Symposium*, vol. 109, 2011.

M. R. Hines, U. Deshpande, et K. Gopalan, “Post-copy live migration of virtual machines”, *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, pp. 14–26, Juil. 2009. DOI : 10.1145/1618525.1618528. En ligne : http://doi.acm.org/10.1145/1618525.1618528

P. Hintjens, *ZeroMQ : Messaging for Many Applications*. " O’Reilly Media, Inc.", 2013.

D. Houston et A. Ferdowsi, “Dropbox”, *https :/www. getdrop box. com*, 2008.

K. Jackson, C. Bunch, et E. Sigler, *OpenStack cloud computing cookbook*. Packt Publishing Ltd, 2015.

D. Josephsen, *Building a monitoring infrastructure with Nagios*. Prentice Hall PTR, 2007.

A. Kivity, Y. Kamay, D. Laor, U. Lublin, et A. Liguori, “Kvm : the linux virtual machine monitor”, dans *In Proceedings of the 2007 Ottawa Linux Symposium (OLS’-07*, 2007.

R. Kuć et M. Rogoziński, *Mastering ElasticSearch*. Packt Publishing Ltd, 2013.

A. Kumar et D. Shelley, “Debugging and troubleshooting”, dans *OpenStack Trove*. Springer, 2015, pp. 149–164.

W. Li, “Algorithms and systems for virtual machine scheduling in cloud infrastructures”, Thèse de doctorat, Umeå University, Department of Computing Science, 2014.

F. Ma, F. Liu, et Z. Liu, “Live virtual machine migration based on improved pre-copy approach”, dans *2010 IEEE International Conference on Software Engineering and Service Sciences*, July 2010, pp. 230–233. DOI : 10.1109/ICSESS.2010.5552416

C. C. Marquezan, D. Bruneo, F. Longo, F. Wessling, A. Metzger, et A. Puliafito, “3-d cloud monitoring : Enabling effective cloud infrastructure and application management”, dans *10th International Conference on Network and Service Management (CNSM) and Workshop*, Nov 2014, pp. 55–63. DOI : 10.1109/CNSM.2014.7014141

H. Mi, H. Wang, G. Yin, H. Cai, Q. Zhou, et T. Sun, “Performance problems diagnosis in cloud computing systems by mining request trace logs”, dans *2012 IEEE NOM Symposium*, April 2012, pp. 893–899.

D. Milojičić, I. M. Llorente, et R. S. Montero, “Opennebula : A cloud management tool.” *IEEE Internet Computing*, vol. 15, no. 2, 2011.

J. Montes, A. Sánchez, B. Memishi, M. S. Pérez, et G. Antoniu, “Gmone : A complete approach to cloud monitoring”, *Future Gener. Comput. Syst.*, vol. 29, no. 8, pp. 2026–2040, Oct. 2013. DOI : 10.1016/j.future.2013.02.011. En ligne : http://dx.doi.org/10.1016/j.future.2013.02.011

A. Montplaisir-Gonçalves, N. Ezzati-Jivan, F. Wininger, et M. R. Dagenais, “State history tree : An incremental disk-based data structure for very large interval data”, dans *Social Computing (SocialCom),2013*.

A. Muller et S. Wilson, *Virtualization with Vmware Esx Server*, 1er éd. Syngress Publishing, 2005.

V. Oracle, “Virtualbox user manual”, 2011.

K. Pepple, *Deploying openstack*. " O’Reilly Media, Inc.", 2011.

B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, et S. Shenker, “Extending networking into the virtualization layer.” dans *Hotnets*, 2009.

B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, et M. Casado, “The design and implementation of open vswitch”, dans *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA : USENIX Association, Mai 2015, pp. 117–130. En ligne : https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

A. Reelsen, “Using elasticsearch, logstash and kibana to create realtime dashboards”, *Dostupné z : https ://secure. trifork. com/dl/goto-berlin-2014/GOTO_Night/logstash-kibana-intro. pdf*, 2014.

B. P. Rimal, E. Choi, et I. Lumb, *A Taxonomy, Survey, and Issues of Cloud Computing Ecosystems*. London : Springer London, 2010, pp. 21–46. DOI : 10.1007/978-1-84996-241-4_2

S. Rostedt, “Finding origins of latencies using ftrace”, *Proc. RT Linux WS*, 2009.

W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc., 1998.

M. Tarsel. ovs tracing. En ligne : http://events.linuxfoundation.org/sites/events/files/slides/ovs-trace-points-ONS-2016.pdf

J. Turnbull, *The Logstash Book*. James Turnbull, 2013.

A. van Hoorn, J. Waller, et W. Hasselbring, “Kieker : A framework for application performance monitoring and dynamic software analysis”, dans *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. ACM, April 2012, pp. 247–248. En ligne : `http://eprints.uni-kiel.de/14418/`

A. Videla et J. J. Williams, *RabbitMQ in action*. Manning, 2012.

S. Vinoski, “Advanced message queuing protocol”, *IEEE Internet Computing*, vol. 10, no. 6, pp. 87–89, 11 2006, copyright - Copyright IEEE Computer Society Nov 2006 ; Dernière mise à jour - 2010-06-06 ; CODEN - IESEDJ. En ligne : `http://search.proquest.com/docview/197318680?accountid=12543`

ANNEXE A Analyse du service Nova à travers RabbitMQ

Le projet Nova de OpenStack est composé de services qui communiquent entre eux à travers un canal de messagerie comme RabbitMQ, qui est basé sur le protocole AMQP (Advanced Message Queuing Protocol). Puisque tous les messages passent par ce canal, il est possible de retrouver les opérations exécutées par les services en s'y connectant. Les messages seront collectés comme des évènements et stockés pour une analyse future.
Nous proposons ici une visualisation des traces de RabbitMQ avec Trace Compass. Le principe est basé sur plusieurs étapes :

1. Collecte et stockage des traces de messages - RabbitMQ propose un mécanisme qui permet à chaque service de se connecter comme récepteur sur une *fréquence* donnée pour récupérer les messages propres à un service qui est enregistré comme émetteur sur cette même fréquence.
   Nous utilisons un script qui se connecte sur les fréquences utilisées par Nova pour récupérer ses évènements, les traiter et les stocker au format XML.
2. Analyse des traces dans Trace Compass - Trace Compass permet d'investiguer des traces au format texte ou XML grâce à un parseur et à un script d'analyse pour visualiser les états des opérations. Nous avons défini un parseur et un analyseur pour les évènements de Nova, mais le même mécanisme peut être appliqué aux autres projets d'OpenStack. L'étape de l'analyse consiste à importer le code du parseur et de l'analyseur de traces.

La Figure[A.1] est un exemple de vue obtenue avec les traces de nova. Le code source et les traces utilisées sont disponibles en [Bationo (d)].

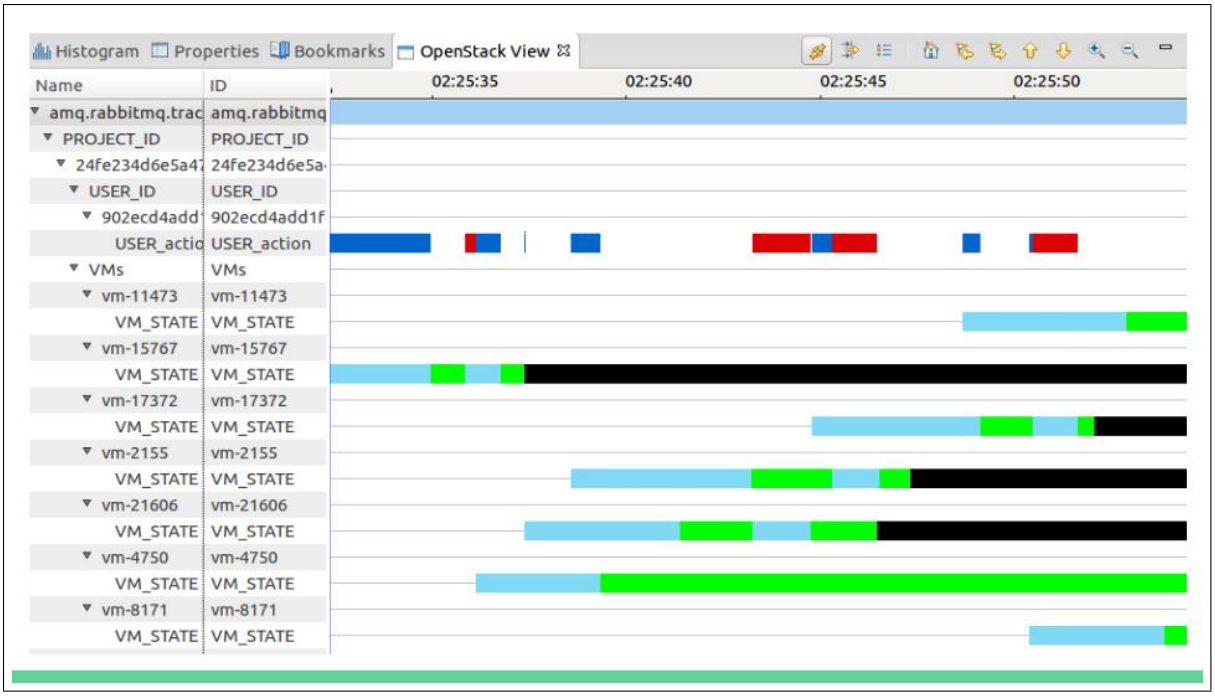

Figure A.1 Exemple de vue Trace Compass avec des traces de RabbitMQ