Improving HPC support in OpenStack by introducing a resource reservation service

François Rossigneux
INRIA, Lyon, France
Email: francois.rossigneux@inria.fr

Jean-Patrick Gelas
University of Lyon, France
Email: jean-patrick.gelas@univ-lyon1.fr

Laurent Lefèvre
LIP Laboratory, ENS Lyon, France
Email: laurent.lefevre@ens-lyon.fr

***Abstract*—OpenStack ne dispose pas d'un système de réservation de ressources dans le temps. Or, cette fonctionnalité est nécessaire pour obtenir de bonnes performances dans le cadre d'un contexte High Performance Computing (HPC). Comme OpenStack ne disposait pas d'un tel mécanisme, nous avons développé un framework de réservation, appelé Climate, qui s'interface avec Nova, l'ordonnanceur d'OpenStack. Climate gère les réservations et l'affectation des hôtes réservés, en tenant compte, entre autres, de leurs performances énergétiques, et fournit un ID de réservation. Cet identifiant de réservation est passé à Nova sous forme de hint, afin qu'il puisse récupérer auprès de Climate les hôtes réservés. Aucune fonctionnalité n'est enlevé de Nova: il est toujours possible de lancer des instances non réservées, simplement en ne fournissant pas d'id de réservation. Les propriétés énergétiques des hôtes sont calculées à partir des relevés effectués par les wattmètres, et remontés grâce à Kwapi (notre framework de remontée énergétique), pondérés ensuite par la performance en calcul des hôtes. En fonction du calendrier de réservation et des performances énergétiques des hôtes, il est possible de prendre des décisions d'extinction des machines inutilisées.**

I. Introduction

Avec l'avènement du cloud et la mutualisation des équipements, les utilisateurs sont amenés à partager la même infrastructure. Cette consolidation apporte des avantages certains en terme d'efficacité énergétique et de souplesse d'utilisation. Cependant, si ce nouveau modèle convient à la plupart des usages, certains d'entre eux, tels que le HPC, sont difficilement transposables dans le cloud à l'heure actuelle. En effet, les applications HPC sont gourmandes en ressources et très sensibles à toute variation de performances entre les noeuds de calcul. Il faut donc avoir une homogénéité des noeuds en terme de performances CPU / GPU, mais aussi de réseau (bande passante, latence). Cette homogénéité évite que les noeuds les plus lents ralentissent les plus rapides. Il faut aussi veiller à ce que l'activité des autres utilisateurs ne perturbe pas les tâches HPC. Pour cela, un utilisateur doit pouvoir réserver des machines, et la portion de réseau utilisée entre elles, afin d'être en environnement single-tenant et éliminer toute perturbation. La réservation de machines physiques, nécessaire dans un contexte HPC, semble aller à l'encontre de la tendance à la consolidation, qui vise à maximiser le taux d'utilisation des ressources. Mais c'est sans compter la spécificité des tâches HPC, qui sont très intensives et accaparent souvent toutes les ressources de calcul des machines. Si la consolidation était possible, elle apporterait non pas des avantages (meilleur taux d'usage et meilleur efficacité énergétique), mais au contraire perturberait la performance des applications HPC, au point de rendre leur exécution dans le cloud impossible. Or, actuellement, les applications HPC sont souvent exécutées sur des infrastructures privées, souvent surdimensionnées, très onéreuses, et rarement exploitées de façon continue à leur plein potentiel.

OpenStack, solution de cloud open source, connait une forte croissance. Nous proposons donc de lui adjoindre un service de réservation, pour supporter les applications HPC et bénéficier des avantages procurés par un tel système.

Dans cet article, nous verrons dans un premier temps comment pourrait fonctionner un tel service de réservation (spécification des besoins, algorithmes de placements, optimisation de certains critères). Nous passerons en revue les avantages procurés à l'utilisateur et au Data Center Infrastructure Manager (DCIM), et nous parlerons des modèles de facturation possibles. Dans un second temps, nous décrirons une proposition d'architecture logicielle, qui viendra se greffer sur OpenStack Nova (scheduler). Nous montrerons aussi comment tirer parti de la présence de wattmètres sur les machines, pour choisir les machines les plus efficientes.

II. État de l'art

Nous verrons comment les utilisateurs peuvent réserver des ressources sur les principales plateformes commerciales (Amazon EC2, Windows Azure, OVH, Grid'5000). Ces ressources peuvent être virtuelles (des instances) ou physiques (des noeuds de calcul). Ensuite, nous ferons le bilan et parlerons d'Haizea, un framework de réservation de ressources compatible avec OpenNebula.

A. Amazon EC2

La plateforme Amazon EC2 propose un système de location de ressources qui convient à beaucoup de cas d'utilisation différents. En la comparant avec d'autres plateformes, et notamment Windows Azure, nous constatons que le modèle de réservation d'Amazon est le plus riche.

Amazon propose quatre modes de réservation : à la demande, réservées, ponctuelles, et dédiées (au sein d'un Virtual Private Cloud).

Les instances à la demande ont un tarif horaire plus élevé que les instances réservées, mais ne nécessitent pas de payement préalable. L'accès au ressources n'est pas garantie

en cas de pic de charge. Ce type d'instance convient aux applications à court terme, qui demandent une quantité de resources variable, et qui ne peuvent pas être interrompues. Ce type d'instances est généralement proposé par les fournisseurs de clouds (Windows Azure, etc).

Les instances réservées demandent à faire un payement préalable (réservation sur un à trois ans). Le taux horaire est ensuite bien plus avantageux. Selon l'utilisation prévue, il est possible de choisir entre plusieurs tarifs préalables, pour ensuite bénéficier de taux horaires plus avantageux). L'accès aux resources est garanti. Ces instances sont parfaites pour les applications stables et prévisibles, ou qui nécessitent une quantité de ressource donnée (récupération après désastre). Les instances réservés peuvent être revendues sur la place de marché Amazon, au prix choisit par le vendeur. Ce type d'instances est aussi très souvent proposé par les fournisseurs de clouds (Windows Azure, etc).

Les instances ponctuelles permettent à l'utilisateur de préciser le coût maximum horaire qu'il est prêt à payer. Il n'y a pas de payement préalable, et les instances sont tuées si le coût horaire dépasse celui précisé par l'utilisateur (en fonction de l'offre et de la demande). Ce type d'instance est adapté aux applications avec des dates de début et de fin flexibles, qui ne sont faisable qu'à très bas coût (ces instances sont moins chères que les instances à la demande), ou qui sont très urgentes (ces instances sont utilisées pour rajouter à la volée de la capacité additionnelle). La facturation se faisant à l'heure, la dernière heure "partielle" n'est pas facturée (à moins que l'utilisateur termine volontairement son instance). À notre connaissance, seul Amazon EC2 supporte cette fonctionnalité.

Enfin, les instances dédiées sont disponibles dans l'offre Virtual Private Cloud (VPC). Le VPC permet de constuire un pont sécurisé entre l'infrastructure locale (d'une entreprise) et l'insfrastructure Amazon, en passant par un VPN. Les instances à l'intérieur de ce VPC sont isolées des autres instances (au niveau réseau). Par défaut (sans instances dédiées), les instances appartenant à un VPC peuvent être exécutées sur des machines partagées entre plusieurs clients. Pour être certain d'être dans un environnement single-tenant, il faut utiliser des instances dédiées. On remarquera que plusieurs instances dédiées appartenant au même client ne sont pas forcément lancées sur la même machine, afin de diminuer l'impact en cas de panne matérielle. Les instances dédiées se rapprochent de la réservation de ressources physiques, à la différence que les caractéristiques de l'hôte physique ne sont pas choisies directement par l'utilisateur.

Dans chacun des modes de réservation, il est possible de lancer des instances standard, et des instances HPC (optimisées pour le calcul, pour la mémoire, pour le stockage, avec GPU, etc). Ces instances optimisées seront lancées sur des machines avec des performances matérielles suffisantes. L'obtention de noeuds avec une faible latence se fait simplement en chargant les instances dans le même groupe d'emplacements. Amazon indique quels CPU / GPU sont associés à ces instances HPC, afin que l'utilisateur optimise ses applications en conséquence.

B. OVH

OVH propose deux offres Dedicated Cloud, qui reposent sur VMware. L'offre vSphere as a service propose un cloud dédié, avec une visibilité sur le matériel, tandis que l'offre vCloud as a service fait abstraction du matériel. Une dizaine de configurations serveurs sont proposées, que l'utilisateur peut choisir avec l'offre vSphere as a service. Dans tous les cas, il peut rajouter des ressources ou en enlever selon ses besoins. La facturation est à l'heure. Ces Dedicated Cloud ne sont pas conçus spécifiquement pour le HPC. Pour obtenir du matériel dédié au HPC, il faut s'adresser à Oxalya, filiale d'OVH.

C. Grid'5000

Grid'5000 est une plateforme d'expérimentation scientifique, qui diffère des solutions commerciales sur de nombreux points. Son infrastructure est publique et l'utilisation gratuite. C'est la seule plateforme parmi celles étudiées ici qui permet de réserver des ressources en avance, et en précisant les caractéristiques des hôtes souhaitées. Toutes les autres plateformes (à l'exception des OVH Dedicated Clouds vCloud as a service) font choisir l'utilisateur parmi une liste de "flavors". En effet, contrairement aux plateformes commerciales qui sont réticentes à dévoiler leurs infrastructures physiques, Grid'5000 fournit beaucoup de détails sur les caractéristiques matérielles des hôtes, car il s'agit d'une plateforme permettant de réaliser des expériences. L'utilisateur doit donc pouvoir choisir avec beaucoup de finesse ses hôtes. Les hôtes réservés sont affectés à l'utilisateur au moment de la réservation. L'utilisateur se retrouve dans un environnement single-tenant (comme sur un instance dédiée Amazon), à la différence qu'il a pu choisir le matériel sous-jacent. Grid'5000 dispose aussi d'un mode best effort, qui ressemble aux instances ponctuelles d'Amazon. La différence est qu'avec Amazon, une instance ponctuelle est interrompue si le prix devient trop élevé, tandis que dans Grid'5000, comme il n'y a pas de facturation, l'instance est tuée si un utilisateur a réservé la machine sur laquelle l'instance best effort est exécutée.

D. Framework Haizea

Il n'est pas possible de réserver des ressources en avance avec OpenStack. À notre connaissance, seul OpenNebula propose ce type de service, s'il est couplé avec Haizea. Haizea est un système de reservation et d'ordonnancement qui supporte différents types de réservations (immédiates, futures et best effort). Il prend en compte les ressources et le temps nécessaire pour préparer la réservation (transfert des images), et celles occupées par les VMs. Ainsi, les réservations peuvent commencer précisément à l'heure requise, et l'utilisateur obtient exactement la quantité de mémoire disponible qu'il a demandé. Il peut être couplé à OpenNebula, ou fonctionner en mode simulation. Haizea est écrit en Python et est sous licence Apache 2, comme OpenStack. Haizea a été créé en décembre 2009.

E. Bilan

Les plateformes commerciales vues jusqu'à présent n'ont pas de calendrier de réservation. Cela n'est pas gênant à condition qu'il y ait suffisamment de ressources pour honorer tous les clients qui veulent des instances réservées. Le problème des ressources monopolisées avant qu'une réservation devienne active est résolu en proposant des ressources de type best effort, moins prioritaires.

Seul Grid'5000 permet de réserver des ressources à l'avance, avec des hôtes attribués au moment de la réservation. En effet, comme les expériences sont courtes, il est envisageable d'en prévoir plusieurs à la suite (tandis qu'avec Amazon, les réservations durent plusieurs années), et comme les ressources sont très hétérogènes, il est probable que deux utilisateurs aient besoin de la même machine, ce qui justifie le besoin d'un calendrier de réservation.

Dans notre contexte qui utilise OpenStack, Haizea semble être une bonne base qui pourrait être adaptée facilement pour correspondre à nos besoins. Nous souhaitons en effet utiliser un calendrier de réservation, pour convenir à différentes tailles d'infrastructures (éventuellement petites et très chargées). Dans la section suivante, nous verrons comment intégrer un système de réservation à OpenStack, basé sur l'ordonnanceur Haizea, et qui permette d'obtenir des ressources adaptées aux applications HPC.

III. Système de réservation

Pour créer des réservations, l'utilisateur doit pouvoir exprimer ses besoins. Ensuite, un algorithme d'ordonnancement trouve les différentes possibilités en fonction de ces besoins et d'autres critères.

A. Expression des prérequis matériels

Les prérequis matériels permettent à l'ordonnanceur de sélectionner les hôtes appropriés.

1) Choix dans une liste: cette méthode est surtout utilisée par les systèmes qui supportent uniquement la réservation d'instances en faisant abstraction du matériel. Ceux-ci n'ont pas besoin de donner à l'utilisateur la possibilité de choisir les caractéristiques de son hôte physique, car cela n'a pas d'importance. L'utilisateur choisit parmi une liste de flavor (CPU, mémoire, disque), et l'ordonnanceur trouve ensuite un hôte capable de supporter cette flavor (OpenStack).

Cette méthode est aussi utilisée par les fournisseurs qui proposent un choix restreint d'hôtes physiques différents (OVH).

2) Construction d'une expression des prérequis: cette méthode est surtout utilisée par les systèmes ouverts (Grid'5000), et dans lesquels les utilisateurs veulent choisir précisément leurs machines. Cependant, OpenStack supporte cette fonctionnalité grâce à des filtres, qui permettent de filtrer sur des propriétés matérielles plus précises, telles que les CPU infos.

L'utilisateur a parfois une idée très précise des caractéristiques matérielles des hôtes qu'il souhaite réserver, et d'autres fois beaucoup moins. Pour établir la liste des caractéristiques souhaitées, il doit pouvoir choisir parmi un catalogue de propriétés (modèle de CPU, taille du cache mais aussi capacité réseau, etc), et composer une expression en utilisant des opérateurs logique. Cependant, dans un contexte HPC, les applications sont sensibles aux disparité de performances entre les noeuds. Il serait nécessaire de pouvoir exprimer ce besoin, sans pour autant donner obligatoirement une valeur. Par exemple, un utilisateur peut vouloir n'importe quel fréquence de processeur, pourvu qu'elles soient toutes les mêmes.

Ce besoin d'homogénéité s'applique généralement aux CPUs et GPUs, et carte réseau, car ces propriétés dégradent la performances des applications distribuées si elles ne sont pas homogènes. Une quantité de mémoire homogène peut également être importante, dans la mesure où cela facilite le travail du développeur, qui optimise son application pour une taille de mémoire donnée. Mais si un noeud a plus de mémoire qu'un autre, l'application n'ira pas plus vite, ce sera juste de la mémoire inutilisée et "gaspillée".

B. Choix des hôtes

Les hôtes physiques élus sont choisis soit au moment de la réservation, soit au dernier moment, lorsqu'un chargement d'instance est demandé (on garantie que lorsque la réservation deviendra active, suffisament de ressources seront disponibles).

1) Choix des hôtes à la réservation: cette stratégie permet une meilleure anticipation du coût énergétique, car on connait précisément sur quels hôtes la réservation va être lancée, et on peut mieux en estimer le coût. De plus, plus l'utilisateur anticipe sa réservation, plus il a de chances d'obtenir des hôtes efficients.

Le désavantage de cette stratégie est son manque de robustesse en cas de panne d'un hôte, entre le moment de la réservation et l'instanciation. Mais il serait possible de palier facilement à cet inconvénient, en sélectionnant dynamiquement un nouvel hôte si l'un d'entre eux venait à défaillir.

2) Choix des hôtes à l'instanciation: cette stratégie est plus robuste en cas de panne d'un hôte, les hôtes étant choisis au dernier moment.

Elle donne les hôtes les plus efficients aux premières demandes d'instanciation (peu importe le moment de la réservation). Cette stratégie peut être assez lourde en calcul, car à chaque demande d'instanciation, il faut prendre une décision de placement complexe.

C. Modes de réservations

Le mode de réservation dépendra du cas d'usage.

Réservations immédiates
: Elles doivent être honorées immédiatement, ou pas du tout. Elles conviennent à des applications urgentes.

Réservations en avance
: Elles doivent commencer et terminer à des heures spécifiques. Cela permet d'être sûr d'avoir bien les ressources au moment voulu.

Réservations best effort
: Elles sont placées dans une file, et honorées dès que possible. On peut imaginer des options : avec

deadline, et avec un tarif dégressif. Elle peuvent être non-préemptables ou préemptables. Dans ce cas, elles peuvent être interrompues s'il y a des réservations plus urgentes (immédiates ou en avance). Une réservation interrompue est soit migrée vers un autre hôte (à chaud ou à froid), soit suspendue puis relancée sur le même hôte, soit tuée (c'est alors à l'application de checkpointer).

Le système supporte aussi les instances non-réservées, sans garantie d'être en environnement single-tenant.

D. Algorithme de réservation

Le rôle d'un tel algorithme est de trouver quels sont les emplacements libres pour placer une réservation. Il prend en entrée les paramètres présentés dans la figure **??** et retourne en sortie des propositions de placement de réservation.

1) Algorithme first-fit: L'algorithme first-fit est basique et rapide. Il se contente de trouver la première période libre, pendant la durée requise. Pour cela, il trie la liste des hôtes candidats par date de disponibilité au plus tôt. Ensuite, il prend le premier hôte de cette liste, et essai d'aggréger autant d'hôtes que nécessaire sur cette période de temps. S'il n'y en a pas assez, il trouve le second hôte disponible au plus tôt, et recommence le processus.

2) Algorithmes avancés: Des algorithmes bien plus évolués peuvent être mis en place, pour optimiser les réservations en fonction du le coût du courant, ou pour lisser la charge, ou encore limiter les cycles d'extinction/rallumage des machines.

Lorsque plusieurs choix sont possibles, le scheduler à deux possibilités : soit il n'en retourne qu'un, ne donnant aucun choix à l'utilisateur, soit il retourne la liste des propositions et l'utilisateur choisit celle qu'il veut. La première approche nécessite que l'utilisateur décrive en amont les critères de son choix (afin que le scheduler sache laquelle choisir). La deuxième solution peut présenter plus d'intérêt pour le DCIM, qui choisit, selon ses propres critères, le meilleur choix de son point de vue. De plus, faire choisir l'utilisateur entre plusieurs possibilités peut conduire à dévoiler un peu l'usage du datacenter.

E. Mécanisme de suggestion

Parfois, il serait intéressant de suggérer à l'utilisateur de modifier ses paramètres un petit peu, soit parce qu'aucune réservation n'est possible avec ses critères, soit parcequ'il apparait nettement plus avantageux de modifier ces critères (par exemple une durée un peu moins longue permettrait d'obtenir des machines beaucoup plus efficientes, ce qui peut arranger à la fois l'utilisateur et son fournisseur).

Cependant, ce mécanisme de suggestion est le moins utilisé sur les plateformes où il serait le plus utile : sur les plaformes fermées, où l'usage interne des resources n'est pas transparent, l'utilisateur fait ses requêtes en aveugle. Rien ne lui indique qu'en modifiant un peu sa requête, il obtiendrait satisfaction. Mais si ces plateformes sont fermées, c'est pour une bonne raison, et faire des suggestions serait dévoiler l'état du datacenter. Cependant, un utilisateur qui voudrait le deviner pourrait faire un grand nombre de requêtes, avec des critères trop sélectifs, et en restreignant petit à petit ses critères, trouver les limites du datacenter. Il faut donc se prémunir de ce type d'attaque. Une tentative de réservation qui échoue ne peut pas être facturée, donc on ne peut pas jouer sur ce levier. En revanche, on peut fixer des limites aux valeurs extrêmes des paramètres, et limiter le nombre de requêtes par seconde.

F. Ajout dynamique de ressources à la réservation

Au moment de la réservation, l'utilisateur précise le nombre la quantité de resources nécessaire. Mais au cours de la réservation, ses besoins peuvent évoluer. Il doit pouvoir ajouter des ressources ou en supprimer, à la volée. Le prix horaire de ces ressources supplémentaire sera plus élevé que les ressources réservées en avance (pour inciter l'utilisateur à estimer au plus juste la quantité de ressources nécessaires).

G. Facturation des réservations

Les réservations, en garantissant un environnement single-tenant, permettent une facturation à l'usage précise et incontestable. En effet, sur les environnements multi-tenant, il est difficile de répartir le coût statique des machines : les clients seuls sur une machine seraient défavorisés par rapport à ceux qui se partagent un même machine. En effet, le coût statique d'une machine étant invariable, plus il y a d'utilisateurs par machine, plus la part de coût statique par utilisateur est bas.

La facturation prend en compte une part statique, qui est facturée au moment de la réservation, et une part dynamique calculée une fois la réservation terminée, et qui varie selon l'énergie consommée (kWh consommés multipliés par le coût du kWh). Cela incite les clients à optimiser leurs programmes, et permet une facturation équitable des clients.

Voici les paramètres qui entrent en jeu dans la facturation :

Mode de réservation
: Les réservations immédiates sont les plus chères, et les best efforts les moins chères. Les réservations en avance pourront avoir un tarif dégressif, par tranche de temps : par exemple la première semaine coûtera un peu plus cher que la seconde semaine, et ainsi de suite.

Performance des machines
: Les machines les plus performantes doivent coûter plus cher, car elles sont plus récentes, plus onéreuses, terminent plus rapidement les tâches soumises et sont très souvent utilisées pour des applications urgentes. Ainsi, pour un même ratio Flops/W, la machine la plus rapide devra coûter plus cher.

Utilisation des ressources
: Grâce à un service de réservation, le DCIM peut faire des prévisions d'usage du datacenter, et ainsi proposer des tarifs plus avantageux durant les heures creuses. Au contraire, durant les pics de charge, le tarif pourra être plus élevé. Cela permet de tenir compte du coût de l'énergie qui peut être variable, afin d'en répercuter le coût sur les utilisateurs. De

plus, lisser la charge et éviter les pics peut permettre un meilleur taux d'usage global du datacenter, et faciliter son implantation dans des zones où la puissance maximum qu'il est possible de tirer est plafonnée.

Cependant, proposer des tarifs variable en fonction du temps dévoile quelle est l'activité du datacenter. Selon le degré de transparence du fournisseur, celui pourra choisir de proposer une variation de tarif à plus ou moins fine granularité, afin de préserver un certain niveau de confidentialité. Par exemple, des tarifs variants heure par heure dévoilent beaucoup de choses sur l'activité du datacenter, tandis que s'ils varient par chaque mois, cela masque de l'information. Cependant, même avec une variable à grosse granularité, il sera possible de dégager des courbes de tendances, qui peuvent être porteuse de beaucoup d'informations. De plus, une grosse granularité aura tendance à favoriser les clients plus aggressifs et pénaliser ceux dont les réservations sont placées au endroits les meilleurs. Faut-il mieux risquer de perdre certains clients aggressifs, ou perdre des clients souples ?

Comme il est très difficile de calculer précisément la durée d'une tâche, le système de réservation devrait permettre à l'utilisateur d'arrêter sa réservation avant sa fin, ou au contraire de l'étendre si possible, s'il s'aperçoit que sa tâche n'est pas finie. Il doit être également possible d'annuler une réservation qui n'a pas débutée. Cependant, une annulation doit entraîner une pénalité pour l'utilisateur, car celui-ci pourrait faire des réservations multiples suivies d'annulations pour tenter de deviner la taille et les caractéristiques de l'infrastructure du datacenter. De plus, une annulation avant la fin d'une réservation dénote une mauvaise estimation de la durée de la réservation. Avant le début de réservation, plus l'utilisateur l'annule tôt, mieux il sera remboursé. Cependant, si l'utilisateur l'annule immédiatement après avoir réservé, le taux de remboursement est inférieur à 100% pour éviter qu'il ne fasse des réservations multiples. Une fois la réservation démarrée, si l'utilisateur l'annule, il est remboursé sur la base de la période qui n'a pas été consommée. Le taux de remboursement est fixe (par exemple 75%). L'utilisateur pourrait récupérer son argent, mais il semble plus judicieux de lui rembourser sous forme de crédits qu'il pourra utiliser lors d'une prochaine réservation.

IV. Architecture de réservation

Dans cette section, nous décrivons notre architecture logicielle telle qu'elle sera implémentée dans un premier temps. Nous utiliserons la stratégie de réservation qui consiste à choisir les hôtes au moment de la réservation.

A. Climate

Notre architecture vise à apporter des fonctionnaliés supplémentaires d'ordonnancement, sans pour autant être intrusif au niveau Nova. Nova se compose d'une interface API, et d'un ensemble de filtres et pondérateurs. Voici quel est le workflow actuellement utilisé dans OpenStack: lorsqu'un utilisateur souhaite instancier une VM, il passe en paramètres un certains nombre de critères (flavor, etc), et l'ordonnanceur Nova commence par filtrer les hôtes pour garder ceux qui correpondent aux besoins de l'utilisateur. Ensuite, Nova pondère les hôtes restants grâce à des coefficients. Il existe un certain nombre de filtres déjà programmés dans Nova, et un pondérateur, qui prend en compte la RAM disponible.

Cette infrastructure nécessiterait, comme nous l'avons dit, l'ajout d'un filtre pour garder les hôtes homogènes. Mais d'autres problèmes se posent qui nous conduisent à externaliser le filtrage des hôtes à l'extérieur de Nova : en effet, pour placer une réservation de façon optimale, il est nécessaire d'avoir une vue sur le calendrier de réservation. Intégrer ceci dans Nova, à travers les filtres ou les pondérateurs personnalisés, ne semble pas être le meilleur design.

1) API: L'API offre les fonctionnalités de gestion des réservations. Pour créer les réservations, les utilisateurs utilisent les paramètres suivants : host_properties (propriétés des hôtes), start_time (date de début au plus tôt), end_time (date de début au plus tard), duration (durée de la réservation), quantity (nombre d'hôtes à réserver). Si l'utilisateur ne précise pas les paramètres start_time et end_time, la réservation est de type immédiate (elle commence maintenant et dure la durée précisée). Sinon, il est possible de planifier une réservation dans le futur. La réservation dans le futur peut permettre aux utilisateurs de s'assurer de bien avoir les resources au moment voulu. On peut aussi imaginer des modèles de facturation différents si l'utilisateur anticipe ses réservations. Dans un premier temps, le paramètre "scheduler" n'est pas disponible à l'utilisateur.

Climate dispose d'un frontend REST API pour la gestion des réservations, qui exploite deux backends (Inventory et Scheduler).

L'API effectue deux étapes pour traiter la requête de création de réservation. Premièrement elle contacte Climate Inventory pour trouver les hôtes qui correspondent aux prérequis précisés dans host_properties, et qui sont disponibles. Deuxièmement, les hôtes filtrés sont passés, ainsi que les paramètres de l'utilisateur, à Climate Scheduler, et celui-ci trouve une période libre où mettre la réservation.

L'API de Climate n'est pas seulement utilisée par les utilisateurs. Nova l'utilise aussi, pour trouver les hôtes non réservés ou attachés à une réservation, ainsi que tous les modules qui auraient besoin de consulter le calendrier des réservations.

Cette API est sécurisée par l'emploi des tokens Keystone. Selon le rôle correspondant à ce token, elle permet ou non de retrouver l'ID des hôtes physiques attachés à un réservation. En effet, il s'agit d'une information plus ou moins confidentielle, cachée aux utilisateurs simples qui n'ont pas un rôle d'administrateur.

2) Inventory: Il s'agit d'un service RPC, interrogé par Climate API pour retrouver les hôtes candidats à une demande de réservation. Les hôtes candidats sont ceux qui matchent les

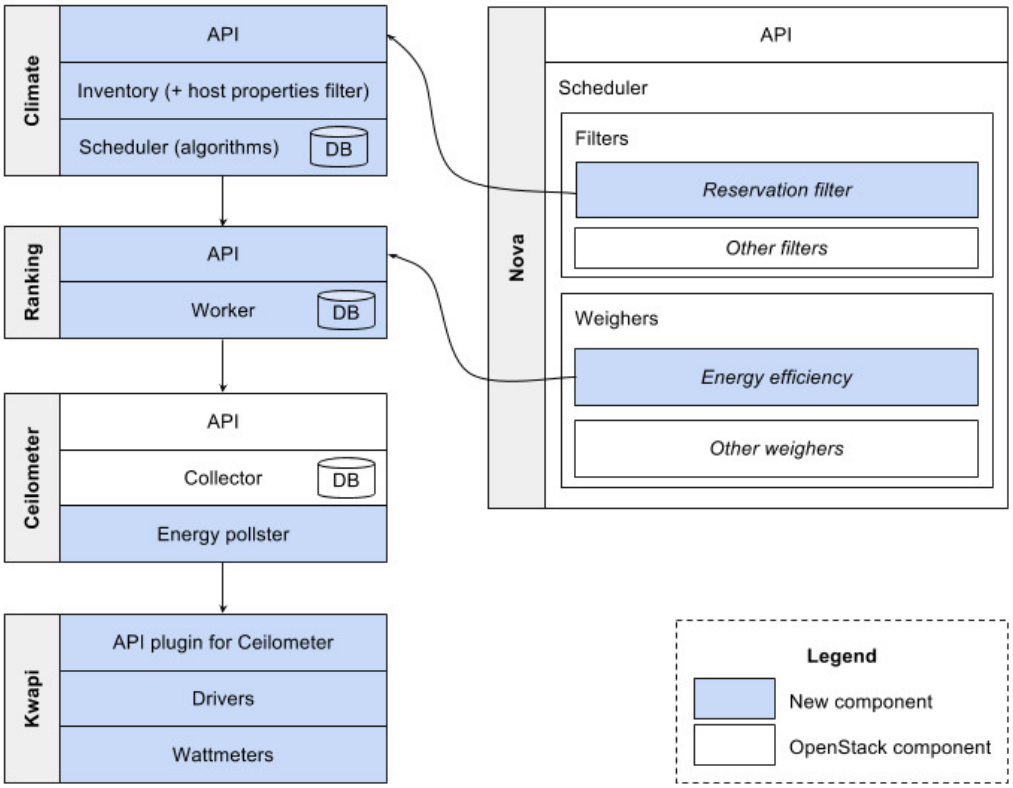

Fig. 1. Global architecture

TABLE I
CLIMATE API

Method	URL	Description
GET	/properties/	Lists the properties
POST	/reservations/	Creates a reservation
GET	/reservations/	Lists the reservations
GET	/reservations/<réservation-id>	Describes a reservation
DELETE	/reservations/<réservation-id>	Cancels a reservation

propriétés requises par l'utilisateur, et ne sont pas utilisés (leur champ running_vm = 0 dans la base Nova). L'API de Nova est interrogée grâce au NovaClient, et le filtrage utilise la syntaxte du json_filter de Nova.

L'inventaire prend en paramètre les host_properties. Mais l'utilisateur doit être capable d'élaborer un tel filtre. Pour cela, il utilise la méthode de l'API /properties/, qui fournit un catalogue des propriétés que l'on peut trier. Cette méthode liste les propriétés présentes dans les détails de hôte enregistrés dans la base Nova, mais le DCIM peut en masquer certaines, et en ajouter d'autres.

B. Nova

Nova est le module de lancement d'instances d'OpenStack. Par défaut, lorsqu'un instance doit être chargée, les hôtes passent dans des filtres qui ne laissent passer que les hôtes éligibles. Ensuite, les hôtes restants sont pondérés, et ceux de poids les plus faibles sont élus. Le scheduler permet de passer des scheduler_hint.

Nova facilite l'intégration de filtre et pondérateurs tiers, en permettant à l'utilisateur de préciser dans le fichier de configuration de Nova le nom du module Python correspondant à l'élément à charger (filtre ou pondérateur). Cette classe hérite d'une classe Nova, et implémente les méthodes nécessaires. L'utilisateur peut passer des hints en arguments, et les filtres récupérerent leur valeur. Ainsi, Climate exploite l'architecture extensible permise par Nova, en fournissant filtres et pondérateurs.

1) Filtering: Le filtre Nova accepte un scheduler hint, nommé réservation_id, qui est l'ID de la réservation retourné par Climate au moment de la réservation. Si l'utilisateur fournit une réservation ID, le filtre contacte Climate API, en fournissant un token admin, et retrouve ainsi la liste des hôtes associés à la réservation. Si l'utilisateur ne fournit pas de réservation ID, Nova contacte Climate pour établir la liste de tous les noeuds réservés à cet instant donné. Seuls les hôtes qui ne sont pas dans cette liste sont éligibles.

2) Weighing: Nous créerons deux Nova pondérateurs.

Le premier pondérera les machines en fonction du temps libre jusqu'à la prochaine réservation : dans le cas d'une instance non réservée, il faudra scheduler cette instance sur la machine qui est libre pendant le plus longtemps, afin de diminuer la probabilité de devoir migrer la VM, si l'utilisateur

n'a toujours par libéré son instance et que l'hôte qu'il utilise va bientôt être réservé par une réservation qui devient active. Dans le cas d'une instance réservée, cette étape est sautée.

Le second pondérera les machines en fonction de leur efficacité énergétique. Ce filtre contactera Kwranking (voir ci-après), en passant la liste des hôtes en paramètre. Il trie ensuite le résultat pour ne garder que les hôtes les plus efficients.

Il faudra veiller à la pondération, afin que le premier filtre ait une "haute priorité".

V. Architecture d'efficience énergétique

L'architecture d'efficience énergétique met à disposition du scheduler les informations nécessaires pour prendre ses décisions de placement.

A. Kwapi

Kwapi is our framework, designed for acquiring energy consumption metrics. It allows, among other, to upload metrics from the wattmeters to Ceilometer.

Its architecture is based on a layer of drivers, responsible for the acquisition metrics, and a layer of plugins that collect these metrics. The communication between these two layers goes through a bus.

Drivers and plugins are easily extensible to support other types of wattmeters, and provide other services.

1) Drivers layer: The drivers are threads started by a manager, which instantiate them with a set of parameters loaded from a configuration file (unified with the OpenStack configuration file format, similar to INI). These parameters are used to query the meters (IP address, port, etc.) and indicate the sensor IDs in the issued metrics. The metrics are Python dictionary with a set of fields. Optional fields can be added, such as voltage, amperage, etc.. The metrics are signed.

The manager periodically checks if all threads are active, and restart them if necessary (incidents may occur, for example if a meter is disconnected or becomes inaccessible). The drivers can manage incidents themselves, but if they finish their execution, it does not matter because they will be automatically restarted by the manager. It is important to avoid losing measurements because the information reported is watts and not kWh: if a value in watts is lost, we lose information.

2) Plugins layer: The plugins retrieve and process the metrics sent by the drivers on the bus. They expose them to other services (Ceilometer) or user (visualization). They can subscribe to all sensors, or just some of them, through a system of prefixes. After checking the message signature, they extract the fields, and process the received data. Kwapi has a plugin API for Ceilometer, which computes the number of kWh of each probe, appends a timestamp, and stores the last value in watts. These data are not stored in a database, as Ceilometer already has one. If a probe has not issued metrics for a long time, the corresponding data are removed. This plugin has a REST API that allows to retrieve the probe names, and W, kWh or timestamp.

B. Kwranking

Pour améliorer l'efficacité énergétique, il faut que les machines les plus efficientes soient utilisées en priorité (en particulier pour les tâches qui demandent beaucoup de puissance).

L'efficacité énergétique se définit en flops/w, donc il faut pondérer les valeurs de consommation récupérées depuis les wattmètres par un indice de performance. Cet indice peut être obtenu à l'aide d'un benchmark, mais cela oblige à exécuter ce benchmark sur les machines après leur installation dans le datacenter. L'autre solution, plus simple à mettre en place, est d'estimer la performance à partir des propriétés CPU (ou GPU) de l'hôte. Cela fait sens dans la mesure où le CPU et le GPU comptent parmi les composants les plus énergivores et dont la puissance demandée est fonction de la charge soumise.

Pour élaborer cet indice, on prend en compte la famille du processeur, le nombre de puces, de coeurs et de threads, ainsi que la mémoire cache et la fréquence. La famille du processeur compte beaucoup, car elle conditionne le jeu d'instruction disponible. C'est d'ailleurs pour cette raison qu'une mesure en Bogomips n'a pas de sens, car cet indice tient compte uniquement du nombre de fois où le processeur est capable de ne rien faire par seconde. Cela ne prend pas en compte le jeu d'instruction plus ou moins optimisé.

Dans le cadre de notre infrastructure énergétique, nous mémorisons, pour chaque hôte, sa consommation min, avg, et max, et l'indice en flop/w. Kwranking est capable d'enrichir une liste hôte avec leurs propriétés énergétique (sans faire de tri toutefois). Cette fonctionnalité est utilisée lors du choix des hôtes, à deux reprises. La première fois au moment de la réservation : par exemple, si l'utilisateur veut des processeurs homogènes, plusieurs familles de processeurs seront candidates, et celles-ci seront pondérées par Kwranking. La seconde fois, au moment de l'instanciation de VMs : d'une part, certaines machines peuvent être plus chaudes que d'autres, et la vitesse des ventilateurs impactent sur la consommation, et d'autre part, des hôtes strictement identiques d'un point de vue matériel peuvent présenter des différences de consommation allant jusqu'à 20% (cela est du aux aléas de la fabrication des puces). Il s'agit donc de choisir, au moment d'une instantiation de VM, quel est l'hôte le moins consommateur parmis les hôtes réservés.

De même, parmi les hôtes réservés, on aura tendance à mettre en veille ceux qui consomment le plus.

L'utilisation de ce module consiste à lui passer une liste d'hôtes en paramètre, qu'il retourne enrichie, pour chaque hôte, de son indice de performances, et de sa consommation minimum, moyenne et maximum.

C. Kwassign

La facturation devrait tenir compte de l'usage qui est fait des ressources. Cela permet d'inciter les clients à mieux optimiser leur programmes et permet à ceux dont les programmes sont légers d'avoir des tarifs plus abordables. Ce modèle de facturation nécessite deux choses : la première est d'avoir des équipements de mesures de la consommation énergétique, et la seconde de pouvoir assigner ces "mesures" à un utilisateur

particulier. Les environnements mono-tenancy se prêtent bien à cette opération, car il n'y a pas besoin de recourir à des estimations de la consommation des VMs, qui pourraient être remises en question. Kwapi, le framework de remontée énergétique, ne connait pas quel utilisateur était assigné à telle machine à un moment donné. Ces valeurs sont stockées dans Ceilometer collector, et il serait souhaitable de les assigner aux clients avant leur stockage, afin qu'ils puissent librement interroger l'API de Ceilometer et retrouver leur consommation énergétique. De plus, cette assignation n'a besoin de se faire qu'une seule fois. Nous avons donc créé un module logiciel qui se branche sur le bus de Ceilometer, et modifie à la volée les métriques remontée, en leur assignant un propriétaire. Pour cela, il interroge le calendrier de réservation de Climate. En revanche se trouve dans Climate le calendrier de réservation. Dans le cas de réservations non réservés, et donc en environnement potentiellement multi-tenancy, on ne réalise aucune assignation, et on ne rend pas non plus accessible la consommation des machines concernées, car cela dévoilerait l'activité des autres utilisateurs.

D. Modes de veille

Climate Power permet de gérer les modes de veille des machines. Comme Climate dispose d'un calendrier de réservation dans le temps, il est facile de s'en servir pour savoir quand mettre en veille et allumer les machines. Pour prendre une décision, il faut tenir compte des prochaines réservation qui vont devenir active, et de la probabilité qu'une instance non réservée soit ordonnancée sur une machine donnée. Lorsque qu'une réservation devient active, soit l'utilisateur a immédiatement besoin de toute ses ressources (dans ce cas là ce module devra anticiper les réservations qui vont devenir active et allumer les machines en avance), soit il veut simplement avoir la garantie de disposer d'un certain nombre de machines, mais n'est pas très regardant sur le délai de mise en route. Dans ce dernier cas, laisser ses machines en veille un peu plus longtemps lui fera économiser la consommation idle. Ou pourrait par exemple donner la possibilité à l'utilisateur de préciser le nombre de machine qu'il souhaite utiliser immédiatement.

L'utilisateur doit avoir accès à l'API de Climate Power pour gérer lui même la consommation de ses machines en fonction du temps, s'il n'a pas besoin de la même quantité de machine dans le temps. Nova aussi utilise cette API, pour allumer un hôte lorsqu'une instance non-réservée doit y être lancée.

Climate doit choisir quel mode de veille utiliser. Plus la veille est profonde, plus cela met du temps à rallumer la machine et plus cela est coûteux en terme d'énergie. On notera qu'il ne semble pas préjudiciable à la durée de vie des machines de les éteindre et rallumer fréquemment (selon REF).

VI. Scénarios

Nous allons évaluer le gain procuré par un système de réservation, pour l'utilisateur et le fournisseur. Le gain en électricité sera utilisé pour fournir des tarifs plus attractifs. En lissant la charge, on peut accueillir plus d'utilisateurs en journée. Avec le plafonnement de la consommation, il est possible de construire plus de datacenters.

Nous prendrons les paramètres suivants :

- La plateforme comporte 1000 noeuds de calcul
- La plateforme est utilisée 12h par jour
- 50% des noeuds consomment 20% de plus que les autres
- Une demande de réservation arrive toute les minutes
- Une réservation réserve en moyenne 10 noeuds
- Une réservation dure en moyenne 1h15
- 33% des utilisateurs sont prêts à décaller leurs requêtes
- Des noeuds à performances homogènes procurent un gain de performances de 33%

a) État initial: initialement, la plateforme utilise OpenStack. Sans réservations et ne voyant pas d'avantages à décaler dans le temps leurs calculs, les clients utilisent la plateforme en journée, de 8h à 20h.

La charge moyenne sur ces 12h est de 0.9 (plateforme proche de la saturation, tandis qu'elle est de 0.45 sur 24h).

La nuit, personne ne bénéficie de l'électricité moins chère, et les machines restent allumées pour rien. En journée, 20% des utilisateurs sont refusés car la plateforme est trop chargée. Une petit pourcentage d'utilisateurs est rebuté par les tarifs élevés (électricité chère et mauvaise optimisation) et s'en va vers la concurrence.

b) Réservations dans le futur: grâce au service de réservation de ressources, 40% des utilisateurs qui réservaient en journée préfèrent réserver la nuit. Par conséquent, la charge en journée passe de 0.9 à 0.54, et de 0 à 0.36 la nuit.

La charge moyenne sur 24h est toujours de 0.45, mais bien mieux répartie. Les utilisateurs ont très peu de chances d'être refusés, ce qui fait que les 20% d'utilisateurs qui étaient partis à cause de la plateforme saturée peuvent revenir. La charge passe donc, de 0.54 à 0.648 en journée, et de 0.36 à 0.423 la nuit.

c) Efficacité énergétique: les requêtes qui saturaient le datacenter le jour sont décallés sur des hôtes beaucoup plus efficients. Étant donné que 50% des noeuds sont 20% plus efficients que les autres, on peut supporter jusqu'à une charge de 0.5 sur des noeuds efficients. La nuit, malgré les 20% nouveaux utilisateurs, toute la charge est absorbée par des hôtes efficients. En journée, seule 0.148 de charge sont traités sur des hôtes moins efficients. À l'origine, 0.4 de charge étaient traités sur des hôtes peu efficients, en journée.

d) Mise en veille: avant la mise en place du système de réservation, les machines ne s'arrêtaient jamais. Comme la charge est de 0.45 sur 24h, on peut éteindre les machines qui ne sont pas utilisées. On peut donc éteindre 55% des machines. Ces machines étant surtout les moins efficientes, on économise une part plus importante que 55% d'énergie. On ignore ici l'énergie requise pour la mise en veille et la sortie de veille.

Grâce à l'économie d'électricité, le DCIM peut proposer des tarifs plus attractifs, attirer de nouveaux clients et augmenter l'utilisation de la plateforme, jusqu'à arriver à nouveau proche de la saturation. Nous voyons donc que la mise en veille est très intéressante pour réaliser des économies, et ensuite attirer

de nouveaux clients. Mais à long, seul un petit pourcentage de la plateforme devrait rester en veille, sinon elle serait surdimensionnée...

Bilan : le bilan final est très positif. Plus aucun utilisateur n'est refusé, car la charge est bien répartie sur 24h, les clients profitent des tarifs heures creuses, beaucoup d'électricité est économisée en éteignant les machines, ce qui permet de diminuer encore les tarifs, et d'attirer de nouveaux clients.

VII. Limitations

Nous pouvons identifier deux types de limitations : celles inhérentes au système de réservation, et celles portant sur des éléments techniques susceptibles d'évoluer.

Il serait souhaitable de donner des conseils de placement à l'utilisateur, pour diminuer sa facture, ou honorer une requête trop contraignante. Cependant, il ne faut pas révéler trop d'éléments sur le datacenter.

S'il n'y a plus assez de machines libres pour honorer une demande de réservation, les instances non réservées pourraient être migrées et aggrégées afin de libérer de la place. Pour cela, il faut regrouper ces instances, en fonction du type d'hôte requis.

VIII. Conclusion

Nous avons vu que dans un contexte HPC, pouvoir avoir la garantie d'être seul sur sa machine était indispensable. Pour cela, il y a besoin d'avoir un système de réservation de ressources. Ce calendrier de réservation des ressources impliqué dans un tel système peut aisément être exploité pour bénéficier d'autres avantages.

D'un point de vue utilisateur, la réservation permet de garantir l'accès au ressources dans le futur. Sans réservation, il y aurait une concurrence accrue entre les utilisateurs, et il ne serait pas possible aux utilisateurs prévoyant d'avoir plus de chances d'accéder aux ressources en anticipant leur réservation.

Pour le DCIM, il est possible de prévoir la charge du datacenter, et de proposer des tarifs qui varient en fonction de cette charge, pour inciter les utilisateurs à réserver durant les périodes creuses. Cela permet donc d'une part un lissage du trafic, et d'autre part on peut imaginer la possiblité pour le DCIM de plafonner la consommation énergétique : en connaissant les réservations à l'avance (en tout cas dans le cas du modèle où les hôtes sont choisis au moment de la réservation) il est possible de savoir s'il peut accepter des nouvelles réservation durant une période donnée, sans dépasser la consommation instantannée maximale qu'il se sera fixée.

D'autre part, le contexte HPC nécessitait d'obtenir des noeuds à performances homogène. Devant la variété des besoins utilisateurs, il est apparu nécessaire de lui donner la possibilité de décrire les caractéristiques de ses hôtes avec une fine granularité. Et selon les hôtes candidats et leurs périodes de disponibilité, le système peut s'appuyer sur des stratégies internes afin d'optimiser certains critères (efficacité énergétique, charge du datacenter, coût du courant électrique, etc). Ces stratégies étant choisies soit par le DCIM, soit par l'utilisateur.

Un tel système de réservation de ressources s'applique ici à la réservations d'hôtes entier, mais il pourrait être généralisé à bien d'autres types de ressources (espace de stockage, réseau, etc).

Acknowledgment

Not yet available.

References

Not yet available.