UFR 919 Sciences de l'ingénieur
Master Informatique – Spécialité STL
ALASCA

Gestion autonomique d'un centre de calcul simulé

Jacques Malenfant
professeur des universités

version 3.0 du 12/09/2017

Résumé

L'objectif de ce projet est de comprendre les possibilités offertes par l'approche autonomique dans la gestion des centres de calcul pour allouer les ressources en fonction d'objectifs comme la performance des applications ou la consommation globale d'énergie du centre.

1 Problématique générale

La virtualisation et l'utilisation de centres de données, les fondements du *cloud computing*, jouent un rôle crucial dans la montée en puissance d'une informatique mondialisée, qui se caractérise par des applications web ouvertes à un grand nombre d'utilisateurs, s'exécutant en continu 24/24, 7/7, le plus souvent avec de grandes fluctuations de leur charge au fil de l'exécution. Parmi différents aspects importants, la possibilité offerte par le *cloud computing* de gérer dynamiquement les ressources allouées aux applications est fondamentale. Non seulement elle permet de n'utiliser que les ressources réellement nécessaires pour chaque application à tout instant, mais aussi elle permet de partager les ressources physiques entre les applications qui n'ont pas les mêmes besoins aux mêmes moments. Qui plus est, elle permet aussi de n'utiliser globalement que les ressources nécessaires et donc de contenir l'impact économique et écologique des centres de calculs en limitant leur consommation d'énergie.

D'un point de vue strictement économique, les centre de calculs louent leurs ressources le plus souvent sous la forme de machines virtuelles. Les clients peuvent louer ces machines virtuelles en fonction de leurs besoins, mais ces derniers fluctuent en fonction de la demande des utilisateurs finaux des applications. Louer un certain nombre de machines virtuelles une fois pour toutes mène le plus souvent soit à mal servir les clients dont la demande excèdera la capacité de calcul des machines virtuelles allouées, soit à des coûts trop importants si la capacité des machines virtuelles excède la demande.

Autre facette économique et écologique, la consommation d'énergie de l'ensemble des centres de calcul dans le monde est évaluée aujourd'hui dans une fourchette entre 3 et 5% de la production d'énergie électrique totale du monde ! Et ces chiffres ont doublé dans les trois dernières années. Par exemple, un seul centre de calcul installé en région parisienne consomme jusqu'à 60 Mégawatts d'électricité, soit la consommation moyenne d'une ville de 60.000 habitants. Utiliser au mieux les centres de calcul est donc un enjeu mondial majeur, et il n'est pas exagéré de dire que les grands opérateurs de centres de calcul ne font des profits que dans la mesure où ils arrivent à gérer leurs ressources informatiques de la manière la plus judicieuse... Écologiquement et politiquement, on se rend vite compte de l'importance de l'impact d'une industrie qui consomme pratiquement l'équivalent de l'ensemble de la production du parc nucléaire français à l'échelle mondiale.

L'objectif de l'UE ALASCA est d'apprendre à créer des applications auto-adaptables dynamiquement, en particulier dans la gestion des ressources qui leurs sont allouées. Le projet doit permettre d'expérimenter ces possiblités, et nous utiliserons le cas des centres de calcul comme

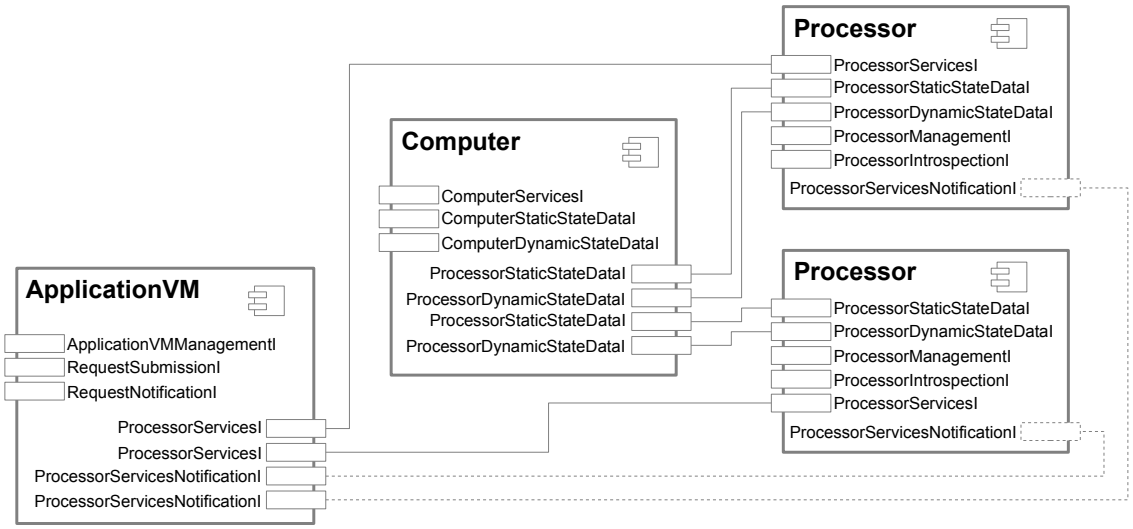

FIGURE 1 – Modèle à composants du centre de calcul.

domaine d'application. Naturellement, nous ne diposons pas d'un véritable centre de calcul pour mener de réelles expérimentations. Le projet sera donc fondé sur une simulation de centre de calcul relativement simpliste, puis des contrôleurs seront développés pour assurer une bonne adaptation dynamique. Des expérimentations permettront enfin de prendre conscience des possibilités de l'auto-adaptation dynamique dans ce domaine.

2 Le simulateur de centre de calcul

Le point de départ du projet est un simulateur simplifié d'un centre de calcul qui inclut la simulation des ordinateurs et de leurs processeurs multicœurs. Elle inclut également celle des machines virtuelles. Et pour donner une première vision globale du processus d'exécution, elle propose aussi un générateur de requêtes simple simulant les utilisateurs soumettant leurs requêtes à une application.

2.1 Le centre de calcul et ses ressources physiques

D'un point de vue physique, un centre de calcul est constitué d'un ensemble d'ordinateurs (ses ressources physiques) qui, pour les besoins du projet, seront simplement vus comme des processeurs multicœurs. Un processeur multicœur a n cœurs (considérons actuellement 2, 4, 8 et 16 cœurs), chacun pouvant fonctionner à différents niveaux de fréquence (prenons encore actuellement 1,5, 2,0, 2,5 et 3,0 GHz). Pour chaque niveau de fréquence, un cœur peut exécuter un certain nombre d'instructions par seconde et a une consommation d'énergie donnée. Les niveaux de fréquence des différents cœurs peuvent être réglés avec la contrainte physique que jamais deux cœurs d'un même processeur peuvent être réglés à des fréquences différant de plus d'un certain écart, par exemple de 0,5 GHz.

La figure 1 présente le modèle à composants du simulateur des ordinateurs et des machines virtuelles. Côté ordinateurs, il y a deux types de composants, les ordinateurs eux-mêmes et les processeurs. Un composant `Processor` offre une interface de services `ProcessorServicesI`, dont l'exécution de tâches sur un de ses cœurs, et une interface de notification de la fin d'exécution d'une tâche `ProcessorServicesNotificationI`. Il offre également des interfaces capteurs `ProcessorStaticStateDataI` et `ProcessorDynamicStateDataI`, de type « data interfaces » fournissant les données en mode « push », de même que des interfaces actionneurs `ProcessorManagementI` permettant de le gérer et de l'adapter dynamiquement. Un composant `Computer` offre également

une interface de ses services de base `ComputerServicesI` et des interfaces capteur `ComputerStaticStateDataI` et `ComputerDynamicStateDataI`. Il reçoit de ses processeurs les informations d'état statiques et dynamiques via des interfaces requises correspondant donc aux interfaces capteurs de ces derniers. La description détaillée de ces interfaces est donnée dans la documentation `javadoc` du simulateur.

2.2 Les machines virtuelles

Les applications ne sont pas exécutées directement sur les machines physiques mais plutôt sur des machines virtuelles qui seront elles-mêmes allouées sur des machines physiques. Pour les besoins du projet, les machines virtuelles vont toujours être déployables à l'intérieur des ressources d'un seul ordinateur physique (pas de MV recouvrant plusieurs machines physiques). Lors de son déploiement, une machine virtuelle se voit allouer un certain nombre de cœurs du processeur de l'ordinateur physique, et elle a un accès exclusif à ces cœurs pour exécuter son application.

Dans le contexte du projet, une machine virtuelle exécute une et une seule application. Une machine virtuelle peut être allouée à une application puis, plus tard, désallouée mais pas directement changée d'application. Si elle est désallouée, ou si l'application se termine, alors la machine virtuelle est réinitialisée pour servir à nouveau ou détruite.

2.3 Les applications interactives et la simulation de leur exécution

Le projet va se concentrer sur les applications interactives, aussi appelées applications *web*. Par application interactive, nous entendons une application qui s'exécute aussi longtemps qu'elle n'a pas été arrêtée volontairement (il n'y a pas de fin intrinsèque à son exécution autrement que par son arrêt explicite, demandée par son fournisseur de service).

Pour les besoins du projet, nous n'allons cependant pas disposer d'applications *web* réelles. Nous allons plutôt utiliser une simulation d'applications qui va s'intéresser uniquement à son aspect consommation de ressources, c'est-à-dire dans notre contexte simplifié le nombre d'instructions à exécuter pour compléter chaque requête. Ce nombre d'instructions va ensuite permettre de connaître le temps d'exécution de la requête en fonction du cœur de processeur utilisé et de sa fréquence courante. Ainsi, à partir d'une caractérisation du nombre d'instructions nécessaires pour compléter les requêtes, la simulation va consister à occuper un cœur de processeur le temps normalement nécessaire pour exécuter ces instructions. Nous revenons plus en détails sur le principe de cette simulation à la prochaine sous-section.

Pour passer d'un nombre d'instructions par requêtes à un temps d'exécution, nous allons supposer que typiquement les applications *web* exécutent les requêtes à l'aide d'un groupe (*pool*) de fils d'exécution (*threads*) attribués à une machine virtuelle. Dans notre contexte, chaque machine virtuelle va disposer d'un fil pour chaque cœur alloué à la machine virtuelle. Voici les étapes de traitement d'une requête par une machine virtuelle :

1. Une requête arrive à la machine virtuelle de l'application, une instance d'application puisque plus tard il sera possible d'en avoir plus d'une pour une application donnée.
2. L'instance d'application regarde si elle a un fil d'exécution en attente et si c'est le cas, lui soumet directement la requête, sinon elle la place dans la file d'attente.
3. Lorsqu'un fil (et donc un cœur) débute l'exécution d'une requête, il l'exécute jusqu'à son terme. Lorsque la requête a fini d'être traitée, le fil demande une nouvelle requête à la file d'attente de la machine virtuelle. S'il y a une requête dans la file, elle est retirée et attribuée au fil qui commence son exécution. Sinon, le fil repasse dans l'état « en attente ».

Le temps de service d'une requête, qui va être la base d'évaluation de la performance de l'application, est la somme du temps passé dans la file d'attente et du temps pris pour son exécution. L'objectif du contrôle de la performance qui va être introduit à l'étape 2 consistera à fixer une valeur cible pour la durée de service moyenne des requêtes et d'ajouter ou retirer des machines

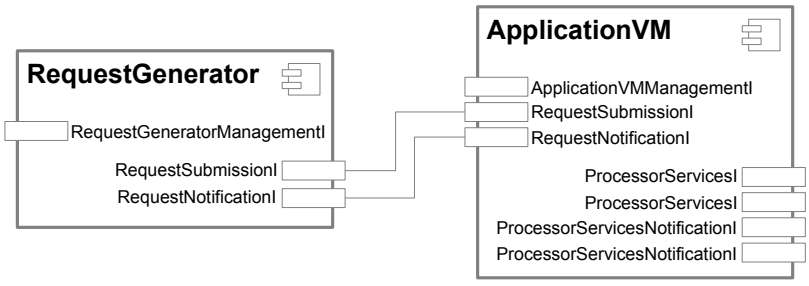

FIGURE 2 – Connexion du client (générateur de requêtes) avec une machine virtuelle.

virtuelles selon que la durée est respectivement plus longue ou plus courte que cette valeur cible. Nous y reviendrons.

2.4 Principe de simulation

Dans le contexte technologique d'ajourd'hui, les applications *web* sont le plus souvent liées à un serveur *HTTP* qui utilise cette norme pour décrire les requêtes. L'application déclenche un traitement à partir de chaque requête, traitement qui peut faire appel à du code PHP ou autre. Une expérimentation réelle supposerait de travailler sur une application et un serveur *HTTP* réels. Une telle approche poserait de grosses difficultés supplémentaires pour mener à bien les expérimentations, difficultés que nous avons voulu (vous) éviter. C'est la raison pour laquelle les applications et le centre de calcul vont être simulés plutôt que de convertir en application autonomique des applications *web* existantes.

Chaque élément du centre de calcul, de même que les applications, vont être implantés comme des composants qui vont posséder des méthodes réalistes (au sens où on pourrait trouver les mêmes dans la réalité) pour recevoir les demandes d'exécution d'applications, la gestion des machines virtuelles, la soumission des requêtes aux applications et le traitement de ces requêtes sur les instance d'applications. Par contre, c'est dans l'implantation du fonctionnement des applications ainsi que dans le fonctionnement des ordinateurs que le fonctionnemment réel sera remplacé par un fonctionnement simulé.

Nous utilisons pour cela les principes de la simulation par événements discrets en temps réel. Pour l'arrivée des requêtes, des composants de simulation extérieurs, appelés `RequestGenerator`, sont ajoutés pour générer des événements d'arrivée de requêtes selon des lois de probabilités à définir et paramétrer. Ces composants extérieurs vont en fait simuler le comportement des utilisateurs et sont connectés dans un premier temps directement à une machine virtuelle chargée de les exécuter. C'est en jouant sur les paramètres de génération aléatoire des événements qu'il sera possible de construire des scénarios de test pour les expérimentations.

Pour l'exécution même des requêtes, plutôt que du code réel à exécuter, puisque nous n'avons pas réellement d'applications, le principe de simulation suppose qu'on peut générer à l'arrivée de la requête un nombre d'instructions fictives qu'elle va nécessiter d'exécuter. Par exemple, à l'arrivée d'une requête, si on suppose que son nombre d'instructions à exécuter suit une loi normale de moyenne 50×10^6 instructions avec un écart-type de 20×10^6, on peut utiliser un générateur de nombre aléatoires (la bibliothèque `commons-math` en fournit d'excellents) pour générer un nombre d'instructions suivant cette loi, par exemple 55×10^6. Lors du lancement de l'exécution, le fil concerné va calculer une durée d'exécution en divisant le nombre d'instructions à exécuter par le nombre d'instructions par seconde exécutables sur le cœur de processeur alloué en fonction de sa fréquence courante. Par exemple, si le cœur alloué est à la fréquence 1,5 GHz et que cela donne une capacité de traitement de 10^6 instructions par seconde, une requête nécessitant 55×10^6 instructions s'exécutera en 55 secondes. Ce calcul fait, la méthode d'exécution simulée des requêtes planifiera (au sens de la simulation par événements discrets) l'événement de fin d'exécution de la requête au temps $t_c + 55$ secondes, où t_c est le temps courant.

3 Étape 1 — Répartiteur de requêtes et contrôleur d'admission

La première étape du projet va consister à compléter le simulateur présenté à la section précédente par l'ajout de répartiteurs de requêtes entre les machines virtuelles allouées à une applicaiton et un contrôleur d'admission gérant l'ajout de nouvelles applications et l'allocation initiale ressrouces à ces dernières.

3.1 Les répartiteurs de requêtes

Dans la description précédente, nous avons décrit un cas simplifié où une application est exécutée par une seule machine virtuelle, et donc il suffit pour le générateur de requêtes de soumettre les requêtes directement à la mchine virtuelle. En réalité, les applications sont généralement exécutées par plusieurs machines virtuelles qui vont se partager les requêtes. Ce partage est réalisé par un répartiteur de requêtes qui s'interpose entre le générateur de requêtes, reçoit toutes ses requêtes et les répartit entre les machine virtuelles allouées à cette application. À la réception d'une requête, le répartiteur parcourt la liste des machines virtuelles dont il dispose et soumet la requête à une des instances (par exemple, la moins récente à en avoir reçu une).

Une politique de répartition simple consiste à supposer que sur chaque machine virtuelle, l'instance de l'application possède une file d'attente des requêtes devant être exécutées, et le répartiteur envoie les requêtes à tour de rôle à chaque machine virtuelle. L'allocation d'une machine virtuelle à une application suppose donc la création d'une instance de l'application dans cette machine virtuelle puis l'inscription de cette instance auprès du répartiteur. La déallocation d'une machine virtuelle suppose l'arrêt de l'envoi de requêtes à cette instance d'application, le traitement des dernières requêtes dans la file d'attente puis la désinscription auprès du répartiteur et la réinitialisation ou la destruction de la machine virtuelle.

Notons que cette organisation est compatible avec les protocoles standards du *web* comme le traitement des requêtes *HTTP* ; on peut donc voir ces éléments comme des bouts de programmes qui emballe les instances d'application et leurs transmettent les requêtes *HTTP* reçues via un serveur *web* comme Apache. Il s'agit donc d'une organisation réaliste pour un centre de calcul.

3.2 Le contrôleur d'admission

D'un point de vue logique, et logiciel, un centre de calcul offre un service d'hébergement d'applications via un *contrôleur d'admission* responsable de recevoir les demandes d'exécution d'applications. Ce contrôleur d'admission peut rejeter la demande s'il n'a pas les ressources nécessaires, mais s'il l'accepte, il localise un ordinateur susceptible de l'exécuter et initie l'exécution sur ce dernier en créant une machine virtuelle, en l'initialisant et en lançant l'exécution de l'application sur cette dernière.

Le contrôleur d'admission propose donc une interface de soumission des applications, et maintient des informations sur l'utilisation courante des ordinateurs du centre lui permettant de faire cette allocation.

3.3 Génération automatique du code

Si on suppose que les demandes d'exécution d'applications comporte l'interface de soumission des requêtes à l'application, il devient possible à partir de cette interface de générer le code des répartiteurs de requêtes et de compléter le code des machines virtuelles qui vont recevoir ces requêtes pour cette application. Pour cela, il faut utiliser l'outil Javassist pour générer le code et l'utiliser pour engendre les répartiteurs et les machines virtuelles.

3.4 Modalité de réalisation

À titre de suggestion, la réalisation de cette étape peut elle-même se décomposer en trois temps :

1. Dans un premier temps, concentrez-vous sur la simulation de l'exécution des applications en programmant manuellement les fonctionnalités correpondant aux répartiteurs de requêtes ; contentez-vous d'une exécution mono-machine virtuelle Java et de créer, interconnecter et lancer tous vos composants statiquement par votre classe de déploiement (similaire à la classe `CVM` dans les exemples fournis avec le modèle de composants).
2. Dans un second temps, toujours en exécution mono-machine virtuelle Java, intégrez les fonctionnalités du contrôleur d'admission, ce qui vous demandera de ne plus créer les machines virtuelles et les répartiteurs de requêtes, ni de les connecter entre eux et avec les générateurs de requêtes *statiquement*, mais plutôt dynamiquement, dans le processus de traitement des demandes d'exécution d'applications au contrôleur d'admission.
3. Dans un troisième temps, vous passerez à la génération dynamique du code des répartiteurs de requêtes et des machines virtuelles pour tenir compte de l'interface de soumission donnée lors de la demande d'admission de l'application.
4. Enfin, dans un quatrième temps, passez à une version pluri-machine virtuelle Java, ce qui va vous demandez d'utiliser à la fois les fonctionnalités de répartition des composants sur plusieurs machines virtuelles avec RMI et celles de création et connexion dynamique de composants entre machines virtuelles fournies par le modèle à composant.

À l'issue de cette étape, vos réalisations devraient permettre :

— d'exécuter plusieurs (au moins deux) applications, chacune ayant son générateur de requêtes, sa répartition de requêtes et plusieurs (au moins deux) machines virtuelles ;
— de soumettre des applications (au moins deux) au centre de calcul et de voir s'exécuter sur ces applications une séquence de requêtes (au moins une dizaine de requêtes par application) ;
— de faire une démonstration à la fois en mono-JVM et en multi-JVM.

4 Étape 2 — Contrôle local de la performance

La deuxième étape du projet va consister à ajouter un contrôleur de performance, c'est-à-dire une contrôleur autonomique, à chaque application de manière à maintenir cette dernière autour d'une valeur-cible requise par le fournisseur de service lors de sa soumission au centre de calcul.

4.1 Instrumentation des entités gérées

Pour exercer un contrôle, il faut instrumenter les éléments gérés de manière d'une part à récupérer les informations sur leur état courant, ce sont les capteurs, et d'autre part d'exercer les actions d'adaptation, ce sont les actionneurs. Cette instrumentation réalisée, il faut définir les interfaces capteurs et actionneurs correspondantes.

4.1.1 Informations sur l'état des applications et des ressources

La première information, cruciale, qui vient à l'esprit est bien entendu la performance de cette application. Nous avons indiqué que la mesure de performance visée est le temps moyen de service des requêtes. Il faut donc instrumenter les applications de manière à obtenir cette information. Instrumenter les applications soumises par les fournisseurs de service n'est pas toujours possible. Il est plus prudent de s'appuyer sur les éléments logiciels fournis par le centre de calcul, dont le répartiteur de requêtes.

Pour recueillir les données de base, le temps de service de chaque requête, une approche consiste à noter le temps d'arrivée de la requête au moment de sa soumission au répartiteur, et calculer la durée lors de la réponse qui va repasser par le répartiteur avant de retourner à l'appelant. Mais puisqu'il s'agit de calculer une moyenne, il faut aussi concevoir et spécifier précisément le calcul de moyenne à effectuer en s'aidant des éléments de réflexion proposés dans le cours magistral à ce sujet.

L'information sur la performance est cruciale mais pas suffisante pour exercer un contrôle : il faut aussi connaître toutes les informations pertinentes sur les machines virtuelles allouées, les machines physiques sur lesquelles elles s'exécutent, dont en particulier les cœurs qui leurs sont affectés, et les fréquences courantes de ces cœurs. Il faut donc ceoncevoir les capteurs et interfaces nécessaires pour recueillir ces informations, en vous appuyant sur les interfaces déjà proposées par les composants `Processor` et `Computer`.

4.2 Adaptations offertes pour influer sur la performance

Les adaptations possibles pour modifier la performance d'une application sont :

— ajouter ou retrancher des machines virtuelles à l'application,
— modifier la fréquence d'un ou plusieurs cœurs d'un processeur physique, et
— ajouter ou retrancher des cœurs à une machine virtuelle.

La mise en œuvre de ces adaptations demandent une coopération avec les mécanismes d'allocation et de gestion des machines virtuelles du contrôleur d'admission d'une part, et une coopération entre les composants simulant les ordinateurs et ceux simulant les applications.

4.3 Un contrôleur à politique prédéfinie

Une fois les répartiteurs de requêtes instrumentés et dotés des interfaces capteurs et actionneurs requis, il faut développer les contrôleurs autonomiques qui vont prendre les décisions d'adaptation. Commencez par implanter des contrôleurs à politique prédéfinie, c'est-à-dire dont les décisions suivent une politique donnée lors de leur création. Parmi ces politiques, une famile très utilisée en *cloud computing* consiste à fixer deux seuils : un seuil inférieur qui, s'il est franchi vers le bas, déclenche l'allocation de nouvelles machines virtuelles, et un seuil supérieur qui, s'il est franchi vers le haut, déclenche la désallocation de machines virtuelles. Dans ce cas, à la création du contrôleur autonomique, les deux valeurs de seuil et le nombre de machines virtuelles à allouer ou désallouer, sont définis une fois pour toute.

À cette étape du projet, vous vous contenterez d'introduire un contrôleur unique et indépendant lors de l'admission de chacune des applications. Toute la coordination nécessaire pour obtenir des machines virtuelles supplémentaires à allouer ou encore pour rendre des machines virtuelles sera réalisée en faisant coopérer chaque contrôleur autonomique directement avec le contrôleur d'admission du centre.

4.4 Modalités de réalisation

Pour cette seconde étape, il vous est fortement suggéré de continuer à utiliser une approche progressive dans la mise au point de votre programme. Voici des jalons qui peuvent vous guider pour ce faire, en prenant soin de bien tester et vérifier le bon fonctionnement.de chaque ajout avant de passer au suivant :

1. Dans un premier temps, intégrez le calcul du temps moyens mis par les requêtes pour être servies et les interfaces capteurs associés.
2. Dans un second temps, implantez les actionneurs qui permettront de modifier la fréquence des cœurs et le nombre de cœurs dans une machine virtuelle.

3. Dans un troisième temps, implantez les actionneurs permettant d'ajouter et de retrancher des machines virtuelles à une application.
4. Enfin, implantez le gestionnaire autonomique qui va mettre en œuvre le contrôle sur les ressources en fonction de la durée moyenne de service des requêtes.

À l'issue de cette étape, vos réalisations devraient permettre :

— de créer une application avec un certain nombre de machines virtuelles et exécuter un flôt de requêtes où vous ferez varier le taux d'arrivée pour mettre en évidence la capacité d'auto-adaptation ;
— de faire s'exécuter des scénarios où la variation du taux d'arrivée des requêtes va faire varier la durée moyenne de leur service de telle manière qu'on voit se déclencher les différents mécanismes d'adaptation que vous aves mis en œuvre : augmentation et diminution des fréquences des cœurs, augmention et diminution du nombre de cœurs dans les machines virtuelles de même que l'augmentation et la diminution du nombre de machines virtuelles pour l'application.

5 Étape 3 — Coordination des contrôleurs

La troisième étape du projet vise à mettre en coordination les différents contrôleurs pour arriver à obtenir des adaptations correctes en présence de multiples applications et donc de mutiples contrôleurs jusqu'ici conçus de manière totalement indépendante les uns des autres. Il existe plusieurs approches pour la coordinaiton, certaines s'appliquant bien à petite échelle, de quelques contrôleurs, et d'autres à grande échelle. Dans le cadre du projet, les adaptations de la fréquence des cœurs et d'allocation des cœurs aux machines virtuelles de différentes applications se prêtent à de la coordination à petite échelle.

Par contre, les adaptations concernant l'allocation des machines virtuelles et de leur utilisation des ordinateurs se déroulent à l'échelle du centre de calcul et donc potentiellement de plusieurs milliers voire dizaines de milliers d'applications, de machines virtuelles et d'ordinateurs. Dans ce contexte, la cordination centralisée typique dans la coordination à petite échelle, ne peut passer à de telles échelles. Ainsi, faire une gestion centralisée des ressrouces du centre par une entité à laquelle tous les contrôleurs doivent s'adresser pour obtenir ou rendre les machines virtuelles va devenir un goulot d'étranglement lorsque le nombre de contrôleurs augmente.

L'application d'une approche décentralisée doit permettre une adaptation globale au niveau du centre de calcul qui émerge d'échanges directs entre contrôleurs autonomiques sans passer par une entité centrale. Ces approches permettent un passage à l'échelle, donc en théorie qui serait applicables aux centres de calcul de grande taille (plus de 10.000 ordinateurs et applications, par exemple), ce qui devient rapidement impossible pour les solutions centralisées.

5.1 Coordinations à petite échelle

Jusqu'ici, nous avons volontairement considéré que la modification de la fréquence des cœurs de même que la modification du nombre de cœurs affectés aux machines virtuelles pouvaient se réaliser de manière autonome au niveau des contrôleurs de chaque application. Malheureusement, les choses ne sont pas si simples.

Pour la fréquence des cœurs, rappelons qu'il existe une contrainte physique sur les processeurs qui font que les fréquences de ses cœurs ne peuvent prendre des valeurs arbitraires car les différences de fréquences sont contraintes. Considérons le cas où deux applications A et B possèdent des machines virtuelles partageant les cœurs d'un même processeur. Si A souhaitent relever la fréquence de ses cœurs mais que B au contraire veut les diminuer, il faut arriver à résoudre ce conflit. Pour cela, on peut créer un arbitre par processeur qui va coordonner les demandes de modifications des fréquences de ses cœurs. On peut imaginer plusieurs politiques de coordination. Par exemple, on peut supposer que si une application demande de relever la fréquence d'un ou plusieurs cœurs mais

que cela crée une différence trop grande, le coordinateur accepte temporairement une différence trop importante, mais impose aux autres contrôleurs des autres applications ayant des cœurs sur le même processur de modifier leurs fréquences dans le même sens dès leur prochaine décision.

Un problème similaire se présente pour l'allocation des cœurs aux machines virtuelles. Rappelons ici aussi qu'il y a une contrainte imposée qui dit que tous les cœurs d'une machine virtuelle doivent se trouver sur un même ordinateur. Lorsqu'une machine virtuelle souhaite avoir un nouveau cœur, elle doit en trouver un libre sur l'ordinateur où elle tourne. Si deux machines virtuelles sur le même ordinateur veulent toutes les deux un cœur, il faut décider laquelle va l'obtenir. On peut ici aussi établir une coordinateur à petite échelle en créant un arbitre au niveau de chaque ordinateur auquel les contrôleurs des applications possédant des machines virtuelles sur cet ordinateur doivent s'adresser pour obtenir (et rendre) les cœurs. Encore une fois, plusieurs politiques de coordination peuvent être appliquées, mais on peut imaginer une politique simple qui consiste à demander un cœur au moment de la lecture des capteurs au cas où le contrôleur en aurait besoin pendant le cycle de contrôle courant, et s'il y en a un disponible, il est réservé à ce contrôleur jusqu'à ce qu'il ait pris sa décision. S'il décide de l'utiliser, il peut confirmer la chose à l'arbitre, sinon il lui rend simplement le cœur qui redevient disponible pour une autre application.

Enfin, la même problématique, se retrouve au niveau du centre de calcul pour l'allocation des machines virtuelles. Si deux applications savent qu'une machine virtuelle est libre, elles peuvent toutes les deux décider de la prendre, mais alors un conflit peut se produire pour l'obtention de cette ressource. On peut imaginer une solution à petite échelle, similaire à celle qu'on vient d'évoquer pour les cœurs, c'est-à-dire que le contrôleur d'admission maintient un groupe de machines virtuelles libres, et au début de chaque cycle de contrôle, les applications en réservent une au cas où elles en auraient besoin, et soi confirment l'utilisation ou la rendent en fin de cycle.

Malheureusement, une telle approche ne passe pas à l'échelle : plus il y aura d'applications, plus il y aura de demandes de réservation faites au contrôleur d'admission qui finira par être submergé si le nombre d'applications augmente trop. Une solution décentralisée est préférable dans ce cas, ce que nous allons maintenant voir.

Pour la coordination à petite échelle, les solutions centralisées sont souvent les plus simples à mettre en œuvre. Deux contrôles se prêtent à de telles soltuions : le contrôle des fréquences des cœurs et le contrôle de l'allocation des cœurs aux machines virtuelles. Dans ces deux cas, il est possible d'associer une entité de coordination à chaque processeur et de faire en sorte que tous les contrôleurs d'applications ayant des machines virtuelles s'exécutant sur une processeur soient liés par une interface de coordination à ce coordonnateur. Chaque modification de la fréquence d'un des cœurs du processeur et chaque modificaiton de l'allocation des cœurs à des machines virtuelles devra passer par ce coordonnateur pour lui permettre d'exercer les politiques de coordination choisies.

5.2 Coordinations à grande échelle

Pour assurer une coordination directe entre les contrôleurs autonomiques, il est nécessaire de les mettre en réseau pair-à-pair. Il s'agit donc de construire un réseau logique incluant tous les contrôleurs autonomiques, permettant donc l'échange entre toutes paires de contrôleurs quelconques. De nombreuses topologies et de nombreux algorithmes pour construire de tels réseaux ont été proposés et étudiés, comme nous le voyons dans le cours magistral, mais pour les besoins du projet, nous allons nous contenter de quelque chose de très simple. Les contrôleurs autonomiques vont simplement être joints par des liens pour former un anneau qui sera géré par le contrôleur d'admission pour les insertions et par les contrôleurs autonomiques eux-mêmes pour les retraits. Les liens du réseau doivent servir à passer les informations de coordination, qui sont décrites dans les deux prochaines sous-sections. Il s'agit donc simplement de connexions normales entre composants, mais prévues pour échanger les données voulues.

La gestion décentralisée des machines virtuelles allouées puis désallouées aux applications peut se faire par la circulation d'identifiants de machines virtuelles libres sur le réseau logique des contrôleurs autonomiques. Un contrôleur souhaitant ajouter une machine virtuelle à son appli-

cation doit attendre d'obtenir un identifiant de machine virtuelle libre pour l'utiliser. Lorsqu'un contrôleur désalloue une machine virtuelle, il en force la réinitialisation puis émet son identifiant sur le réseau logique.

Dans un premier temps, il suffira de supposer que toutes les machines virtuelles sont créées une fois pour toute et leurs identifiants circulent sur le réseau logique tant qu'elles sont libres. Dans un second temps, une solution mixte peut être employée pour diminuer et augmenter le nombre de machines virtuelles disponibles sur le centre en insérant le contrôleur d'admission dans le réseau logique pour lui permettre de voir passer les identifiants de machines virtuelles libres et ainsi décider, en fonction de sa fréquence de réception d'identifiants et de sa gestion de consommation d'énergie, de retirer ou ajouter de nouvelles machines virtuelles.

5.3 Modalités de réalisation

Pour cette troisième et dernière étape, le défi principal sera d'arriver à intégrer différentes solutions sur les fondations préparées auparavant, les contrôleurs de l'étape 2. Plusieurs équipes ayant pris du retard vont devoir compléter avant d'aller de l'avant. L'objectif étant d'étudier la coordination des contrôleurs, il vaudra mieux réalisation une forme de coordination sur un type de contrôle plutôt que de mettre toute l'énergie de l'équipe à compléter tous les contrôles avant de s'intéresser à la coordination.

À l'issue de cette troisième étape, vos réalisations devraient permettre :
— de créer plusieurs applications avec un certain nombre de machines virtuelles initiales se partageant des processeurs du même ordinateur physique et des cœurs de mêmes processeurs (sans un tel partage, la coordination des contrôles de fréquence de cœurs et d'allocation des cœurs deviendrait inutile) ;
— de faire s'exécuter des scénarios où les contrôleurs seront amenés à faire varier fréquences, allocations des cœurs et allocation des machines virtuelles aux applications ;
— de faire apparaître par ces scénarios l'intervention de la coordination entre les contrôleurs ;
— de faire une démonstration à la fois en mono-JVM et en multi-JVM.

6 Réalisations attendues et modalités

Nous vous demandons de réaliser le projet sous Eclipse en prévoyant tout le code nécessaire pour lancer l'application (incluant les bibliothèques) et dans votre rendu vous devrez également joindre des scénarios de test.

Modalités de réalisation et de remise des projets :

— Le projet se fait **obligatoirement** en **équipe de deux étudiants**. Tous les fichiers du projet doivent comporter les noms (balise `authors`) de tous les auteurs en Javadoc. Lors de sa formation, chaque équipe devra se donner un nom et me le transmettre avec les noms des étudiants la formant au plus tard le **29 septembre 2017**.
— Le projet doit être réalisé avec Java SE 7 ou 8. Attention, peu importe le système d'exploitation sur lequel vous travaillez, il faudra que votre projet s'exécute correctement sous Eclipse en Mac Os X (que j'utilise et sur lequel vous devrez me faire votre soutenance finale).
— Votre rendu de projet devra inclure un rapport qui se présentera sous la forme de *documentation Javadoc* des différents paquetages de votre projet, et surtout le *paquetage racine* dont la documentation jouera le premier rôle dans la constitution de ce rapport. La documentation Javadoc sera générée et incluse dans votre livraison dans un répertoire `doc` au même niveau que votre répertoire `src`.
— Le code doit aussi être documenté en Javadoc et commenté. Il doit également être lisible et correctement présenté (indentation, ...).
— L'évaluation comportera quatre épreuves : deux audits intermédiaires, une soutenance à mi-semestre accompagnée d'un rendu de code et une soutenance finale accompagnée du rendu

final du projet et de la documentation associée. Les deux audits intermédiaires dureront 20 minutes (par équipe) ainsi que la soutenance à mi-parcours. La soutenance finale durera 30 minutes, selon les modalités suivantes :

1. Les **deux audits intermédiaires** auront lieu pendant les séances du cours, *a priori* le **25 octobre 2017** puis le **10 janvier 2018**. Ils se dérouleront de manière informelle devant un écran d'ordinateur où vous pourrez me présenter l'état d'avancement de votre projet et répondre à mes questions. Ils donneront lieu à une note valant chacune 10% de la note finale de l'UE.
2. La **soutenance à mi-parcours** portera sur la première étape du projet aura lieu dans la semaine des premiers examens répartis, *a priori* le **22 novembre 2017** ; elle donnera lieu à une note comptant pour 30% de la note finale de l'UE. La soutenance à mi-parcours comportera une présentation des réalisations pendant une dizaine de minutes (devant l'écran sous Eclipse), une courte démonstration de cinq minutes sur un ordinateur fourni (sous Mac Os X) à partir du rendu du projet, et 5 minutes seront réservées aux questions. Les rendus à mi-parcours se feront le **dimanche 19 novembre 2017 à minuit** au plus tard.
3. La **soutenance finale** portant sur l'ensemble du projet mais avec un accent sur les deuxième et troisième étapes aura lieu dans la semaine des seconds examens répartis, *a priori* le **7 février 2018** ; elle donnera lieu à une note comptant pour 50% de la note finale de l'UE. Elle comportera une présentation du rapport d'une dizaine de minutes (en utilisant des transparents), une démonstration d'une quinzaine de minutes sur un ordinateur fourni (sous Mac Os X) à partir du rendu final du projet, et cinq minutes seront réservées aux questions. Les rendus finaux se feront le **dimanche 4 février 2018 à minuit** au plus tard.

— Bien que les audits et la soutenance finale se fassent par équipe, l'évaluation reste **individuelle**. Lors des audits et des soutenances, *chaque étudiant* devrait se montrer capable d'expliquer différentes parties du projet, et selon la qualité de ses explications et de ses réponses, sa note peut être supérieure, égale ou inférieure à celles de l'autre membre de son équipe.

— Les audits et les soutenances auront lieu selon un horaire de passage qui sera publié le moment venu sur le site de l'UE. **Tout retard** de l'équipe de plus de dix minutes à ces soutenances entraînera son **annulation** et une note de 0 sera attribuée aux deux membres de l'équipe pour l'épreuve concernée. Si un des membres d'une équipe arrive avec plus de dix minutes de retard, il sera exclu de l'épreuve et une note de 0 lui sera attribuée. Dans ce cas, si l'autre membre de l'équipe est à l'heure, il passera l'épreuve seul.

— Le rendu à mi-parcours et le rendu final se font sous la forme d'une archive `tgz` si vous travaillez sous Unix ou `zip` si vous travaillez sous Windows que vous m'enverrez à `Jacques.Malenfant@lip6.fr` comme attachement fait proprement avec votre programme de gestion de courrier préféré ou encore par téléchargement avec un lien envoyé par courrier électronique (en lieu et place du fichier). Donnez pour nom au répertoire de projet et à votre archive celui de votre équipe (ex. : équipe LionDeBelfort, répertoire de projet `LionDeBelfort` et archive `LionDeBelfort.tgz`).

— **Tout manquement à ces règles élémentaires entraînera une pénalité dans la note des épreuves concernées !**

Fin du document.