Élasticité de l'exécution des processus métier

Guillaume Rosinosky

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Élasticité de l'exécution des processus métier

THÈSE

présentée et soutenue publiquement le 23 janvier 2019

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Guillaume Rosinosky

Composition du jury

Président :	Parisa Ghodous	professeur à l'Université Lyon I, LIRIS
Rapporteurs :	Walid Gaaloul	professeur à Telecom Sud Paris
	Salima Benbernou	professeur à l'Université Paris Descartes
Examinateur :	Malika Smail	maître de conférences à l'Université de Lorraine, LORIA
Invité :	Duy Tran Quang	manager à Bonitasoft
Encadrants :	François Charoy	professeur à l'Université de Lorraine, LORIA
	Samir Youcef	maître de conférences à l'Université de Lorraine, LORIA

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Je remercie tout d'abord mes directeurs de thèse, François Charoy, et Samir Youcef pour leur disponibilité et leur bon sens. Ils m'ont aidé et motivé, et sans eux je n'aurais pas pu terminer ce manuscrit.

Je remercie ensuite l'ensemble de la société Bonitasoft chez qui j'ai passé plus de 2 ans et demi, et plus particulièrement Jérémy Jacquier-Roux à qui j'ai tellement pris de temps, Duy Quan Trang pour son écoute, Baptiste Mesta, Truc n'Guyen, Charles Souillard ainsi que l'ensemble de l'équipe R&D pour leur expertise. Merci également à l'ANRT, à Bonitasoft, au LORIA et à l'université de Lorraine pour le financement de ma thèse.

Pour finir, je remercie ma compagne Natacha pour son maousse soutien durant cette période difficile, ainsi que Cécile pour ses corrections.

Sommaire

Chapitre 5
Algorithmes d'allocation de ressources et de distribution de tenants

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Depuis l'avènement de l'informatique, les architectures informatiques suivent des alternances de centralisations et de décentralisations. Les mainframes, systèmes initialement utilisés en entreprise à partir des années 60, s'appuyaient sur un serveur central concentrant l'ensemble de la puissance, et sur un ensemble de terminaux passifs. Avec l'augmentation de la puissance des ordinateurs et leur généralisation dans les années 90, cette étape a été suivie d'un mouvement de décentralisation. La puissance et l'intelligence était distribuée entre clients et serveurs, ces derniers s'arrêtant à une synchronisation entre les différents clients. L'arrivée et la démocratisation du Web depuis le milieu des années 90 a de nouveau entraîné un mouvement inverse. La mise à disposition de sites Internet, et l'utilisation de clients légers tels que des navigateurs a renvoyé la gestion des règles métiers au niveau des serveurs.

En effet, pendant des années les logiciels ont été vendus et installés sur les installations des entreprises et particuliers les utilisant. L'avènement des connexions haut-débit ainsi que l'augmentation de la capacité de calcul et de la mémoire ont permis aux logiciels de s'affranchir du besoin d'un support d'installation d'un côté, et de fonctionner dans un navigateur Web de l'autre. Ceci a permis la démocratisation du cloud : avec celui-ci, les utilisateurs peuvent profiter de la capacité de calcul des fournisseurs et n'ont pas à gérer l'infrastructure logicielle et matérielle. Ceux-ci peuvent toucher un public plus large, et ainsi profiter de vastes économies d'échelle en mettant en commun les ressources nécessaires à l'exécution des services. L'utilisation d'infrastructure en tant que service par ces fournisseurs permet d'adapter la quantité et la taille des ressources aux besoins des clients : on parle ici d'élasticité. Toutefois, en pratique ce n'est pas une tâche aisée : les besoins des clients peuvent grandement varier selon l'heure et les jours, ceci amenant de nombreux changements dans les installations. De plus, les applications Web transactionnelles ne passent pas à l'échelle facilement, ceci étant dû aux bases de données relationnelles utilisées pour la persistance des données [3]. Cela pose des limites à la taille des installations, d'autant plus que la mise en commun des ressources et logiciels se généralise.

Les outils BPM (Business Process Management) ne sont pas à l'abri de ce phénomène et de ces limitations. Ces derniers permettent aux entreprises de modéliser leurs processus métiers, facilitant leur automatisation et leurs améliorations, les moteurs BPM permettant leur exécution. L'avènement d'outils de BPMaaS (BPM as a Service - BPM en tant que service) permet aux clients et aux fournisseurs d'exécuter les processus en profitant des avantages de l'architecture du cloud : dématérialisation pour les clients, et mise en commun des ressources pour les fournisseurs. Toutefois, les moteurs BPM sont habituellement des applications Web transactionnelles, limitées dans leur passage à l'échelle. Ce problème provient essentiellement des bases de données relationnelles utilisées pour la persistance des données. Ceci pose des problèmes au niveau de

l'adaptation des ressources.

Le manque de capacité de passage à l'échelle amène à considérer des méthodes alternatives : le déplacement des données d'une installation à une autre. Il s'agit d'une fonctionnalité importante pour l'élasticité. Toutefois, le désavantage de ce type de méthodes est de provoquer des ruptures de service durant le déplacement, ainsi que des chutes de performances pour les autres opérations hébergées sur les bases de données. Nous estimons l'effet de ces migrations en considérant les couches applicatives et persistance, à l'aide d'une approche et d'un framework que nous proposons. A notre connaissance, il n'existe pas de méthode de migration à chaud permettant de déplacer les données sans modification du moteur de base de données et avec des modifications limitées au niveau du logiciel. Nous proposons dans cette thèse une approche de migration à chaud novatrice répondant à ces besoins, que nous appliquons au moteur BPM Bonita. Nous montrons ensuite que les effets des migrations sur les tenants migrés et colocalisés sur les ressources concernées par les migrations ne sont pas à négliger.

Il est important pour les fournisseurs de savoir quand allouer les ressources, et pour quels types, et de distribuer leurs clients sur celles-ci. Cette allocation doit être la meilleur marché possible, tout en respectant la qualité de service négociée avec le client. Le travail présenté dans cette thèse consiste à proposer des méthodes d'élasticité pour fournisseurs de BPMaaS, en considérant coût, qualité de service et migrations. Nous considérons dans ces méthodes les besoins des clients et les capacités des ressources en termes de débit de tâches BPM par seconde. Ceci nécessite d'avoir une estimation de la capacité des ressources dans ces termes. C'est ce que nous proposons avec une autre méthode et un autre framework.

Avec leur architecture basée sur les machines virtuelles, les performances des configurations du cloud ne sont pas toujours stables [4], et peuvent nécessiter de nombreuses itérations. Nos frameworks permettent de réaliser des expérimentations intensives, isolées et répétables à l'aide d'outils Open Source simples à mettre en place. Cette estimation nous permet de tester nos approches sur des ressources de taille réalistes, que nous avons utilisés avec nos méthodes d'élasticité.

Les méthodes d'élasticité que nous proposons nécessitent d'avoir à priori les besoins des clients, les coûts et capacités des ressources et une disposition de départ. Nous proposons trois heuristiques et modèles d'optimisation permettant aux fournisseurs de BPMaaS de réaliser des économies sur leur infrastructure tout en respectant la qualité de service de leurs clients. Celles-ci sont basées sur la considération des migrations à chaud comme critère important, sur un découpage en pas de temps, et sur le débit de tâches BPM comme critère de mesure des clients et des ressources les hébergeant. Nos approches permettent de minimiser le coût complet de l'infrastructure en déterminant la liste des ressources nécessaires, et la distribution des clients sur ces dernières. Dans la première, nous proposons une heuristique dont le but est de trouver l'ensemble des meilleurs coûts et nombres de migrations tous tenants confondus d'un pas de temps au suivant. La deuxième s'applique à trouver un coût intéressant pour un ensemble de pas de temps à l'aide d'une méthode simple à mettre en œuvre, en considérant une limite au nombre de migrations pour chaque tenant. Elle s'appuie sur une version améliorée du premier algorithme, utilisé de manière itérative. La troisième approche est une amélioration de la seconde approche à l'aide d'un algorithme génétique. Elle fournit des coûts plus proches de l'optimum.

Dans la section suivante nous décrivons l'organisation de ce manuscrit.

1.1 Organisation du manuscrit

Le chapitre 2 décrit la problématique de notre travail : quelles méthodes utiliser pour parvenir à économiser les coûts d'infrastructure des BPMaaS en respectant la qualité de service du client ? Les migrations ont-elles des effets sur les tenants hébergés sur les ressources concernées et sur le tenant migré ? Nous présentons un exemple de motivation simple basé sur le cas d'un fournisseur de BPM en tant que service pour illustrer notre problème, et concluons par un résumé de nos contributions.

Le chapitre 3 décrit les concepts de cloud, de BPM et d'élasticité. Nous présentons ensuite l'état de l'art de l'élasticité des solutions SaaS multi-tenant, ainsi que de l'élasticité des solutions BPM.

Le chapitre 4 décrit nos hypothèses : architecture multi-tenant, considération du débit de tâches comme mesure de qualité pour clients et ressources et prise en compte des migrations à chaud. Nous présentons ensuite nos approches et frameworks d'évaluation de la taille des ressources en termes de débit de tâches, et d'évaluation des effets des migrations sur les tenants migrés et colocalisés. Nous décrivons également notre méthode de migration dans cette partie. Les résultats obtenus nous fournissent des bases et justifications solides pour les méthodes que nous décrivons dans le chapitre suivant.

Le chapitre 5 décrit les trois méthodes et modèles que nous avons proposés, basées sur les postulats présentés dans le chapitre métrique : détermination pour un pas de temps de la frontière de Pareto coût et migration, détermination pour plusieurs pas de temps du meilleur coût sous contrainte de migrations, et amélioration à l'aide d'un algorithme génétique.

Nous concluons dans le chapitre 6 en présentant un bilan de nos contributions, leurs limites, ainsi que les perspectives.

Chapitre 2

Problématique et motivations

Dans ce chapitre, nous présentons la problématique de cette thèse en montrant les avantages de l'élasticité. Nous poursuivons une illustration de l'intérêt de l'ajustement des ressources dans les configurations multi-tenant. Nous détaillons ensuite les contributions que nous avons effectuées répondant à cette problématique.

2.1 Problématique

Durant la dernière décennie, des changements majeurs sont intervenus dans la manière dont les entreprises fournissent des logiciels. La généralisation d'Internet, du Web, et des connexions haut-débit ont permis la démocratisation du « Cloud computing », un moyen de profiter de puissance de calcul, de stockage, ou de logiciels à distance. Un des avantages des offres du cloud est leur capacité à fournir des ressources à la demande. Ceci permet aux utilisateurs d'arriver à une gestion plus efficace de leurs ressources. En effet, la possibilité de louer des ressources ou des services « à la demande »permet aux clients de n'utiliser que ceux dont ils ont besoin. Ceci permet une augmentation des capacités de l'infrastructure matérielle et logicielle en cas d'augmentation d'activité, mais également une réduction des coûts engendrés par l'infrastructure matérielle et logicielle quand des ressources ou services ne sont plus nécessaires.

Les fournisseurs de cloud mettent à disposition des clients (entreprises, particuliers et institutions) ces ressources contre rétribution financière. Pour les entreprises et les institutions, le cloud permet d'éviter la nécessité d'héberger et gérer des ressources matérielles ou logicielles ainsi que le besoin de ressources humaines pour gérer cette infrastructure. Les fournisseurs mettent à disposition les ressources nécessaires, à la demande, ou réservée pour une longue durée. Plus généralement, ceci permet de transformer des dépenses habituellement comptabilisées en dépenses d'investissement en dépenses d'exploitation, réduisant les investissements à long terme et l'immobilisation de fonds. Cette approche séduit les entreprises, qui peuvent adapter leurs capacités de traitement à leurs besoins et faire des économies sur leur infrastructure. Ceci est extrêmement intéressant en cas de variations ponctuelles ou saisonnières de l'activité. Dans le cas d'une hausse d'activité, il leur est ainsi possible de louer des services informatiques et de les utiliser en évitant d'acquérir des ressources matérielles et logicielles coûteuses qui ne seraient utiles que durant cette période limitée. Pour les utilisateurs finaux, cette approche permet d'éviter l'installation et la mise à jour de logiciels et de matériel, et de retrouver ses environnements ou logiciels à partir de n'importe quel endroit connecté.

Dans cette thèse, nous prenons le point de vue d'un fournisseur de services logiciels. Celui-ci souhaite mettre à disposition un moteur BPM (Business Process Management) à différents

clients, et fournir ainsi du *BPMaaS*[1] (BPM as a Service). Le souhait du fournisseur *BPMaaS* est de mettre à disposition au moindre coût une solution logicielle hébergée permettant l'exécution des processus métiers d'un client. Pour héberger le moteur BPM, nous partons du principe que le fournisseur de services fait appel à des fournisseurs *IaaS* (Infrastructure as a Service) pour ses ressources matérielles. Comme nous le disions, l'utilisation de ce type de fournisseurs permet d'éviter au fournisseur de services d'acheter et de soutenir son infrastructure matérielle.

Si le prix cumulé de la location de ressources à la demande est souvent supérieur à celui d'une infrastructure de taille et puissance similaire [5], un avantage financier du cloud computing est la possibilité pour les fournisseurs de logiciels de pouvoir utiliser les ressources de manière élastique. Ceci consiste en l'adaptation au mieux de la quantité et la qualité des ressources à la demande des clients finaux qui utilisent le service. Le but est de réduire les coûts au nécessaire. Toutefois, ceci n'est pas aisé pour plusieurs raisons : tout d'abord, le modèle de coût des différents fournisseurs du cloud est complexe à maîtriser, et ensuite les besoins des clients peuvent fortement varier dans le temps.

Concernant le modèle de coût, celui-ci est souvent découpé entre puissance de calcul, stockage et réseau. Ceci consiste par exemple en de nombreuses quantités et capacités différentes en termes de mémoire, famille et nombre de processeurs pour les ressources de calcul, mise à disposition de disques magnétique ou SSD avec bande passante limitée pour le stockage, et facturation à l'utilisation pour le réseau. A ceci, on doit ajouter les nombreuses garanties offertes par les fournisseurs ayant plusieurs niveaux de prix, ainsi que les systèmes de seuils où on peut retrouver différents niveaux de coûts selon la quantité de ressources consommées dans le temps. L'évolution temporelle des demandes d'un client est également un point complexe à maîtriser pour le fournisseur de services. Les besoins en puissance machine, bande passante de stockage ou réseau sont rarement stables, et on peut souvent constater des comportements imprévus et des saisonnalités. Ces dernières peuvent être constatées au niveau de la journée (cycle jour/nuit), de la semaine (week-end), ou de l'année (saisons, soldes, vacances, jours fériés). A ceci il faut ajouter les variations non saisonnières (opérations de promotion par exemple). Par exemple, pour un site d'e-commerce français, les besoins en termes de puissance ne seront pas les mêmes en pleine nuit ou en pleine journée. Le but des mécanismes d'élasticité est de parvenir à suivre ces évolutions en adaptant l'usage des ressources.

La figure 2.1 illustre ce point. On appellera « demande »la puissance machine nécessaire pour que les opérations du client puissent avoir lieu dans une durée raisonnable. On appellera « offre »la capacité fournie en termes de ressources matérielles par le fournisseur de services pour satisfaire les besoins du client. Le fournisseur doit payer ces ressources matérielles. Quand les besoins des clients (la demande) sont supérieurs à l'offre, le système est en sous-capacité. Ceci signifie que la puissance fournie n'est pas suffisante, et que les temps de réponse des logiciels hébergés seront plus élevés et non acceptables pour le client. Pour garantir le temps de réponse prévu pour ses clients, le fournisseur de services doit mettre à disposition plus de ressources matérielles. Un système capable de supporter cette montée en charge est dit « scalable »(figure de gauche). Le problème est que la puissance est onéreuse, celle-ci étant louée à la demande à des fournisseurs d'infrastructure. Un système équilibré et bon marché adaptera la puissance allouée aux besoins, comme on peut le retrouver dans l'exemple de système dit « élastique »(figure de droite). Quand l'offre est supérieure à la demande, les performances du logiciel sont suffisantes pour le client et le système est en surcapacité. Minimiser ces périodes de surcapacité est primordial pour contrôler le prix de l'infrastructure nécessaire. Notre but est de minimiser le coût total, tout en minimisant (ou du moins contrôlant) les périodes de sous-capacité.

1. Parfois nommé *BPaaS* (Business Process as a Service)

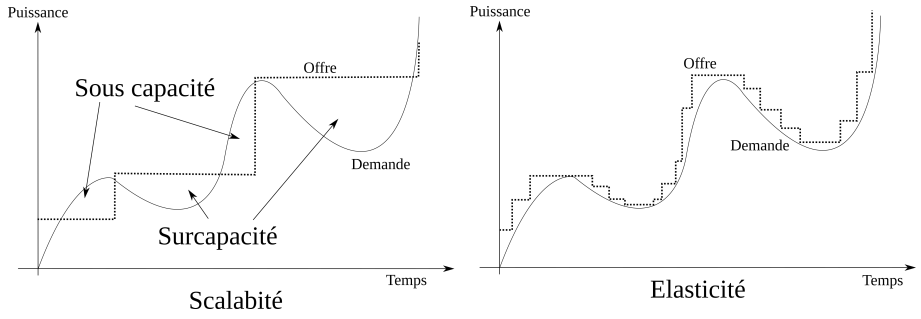

FIGURE 2.1 – Illustration de la scalabilité et de l'élasticité

Il existe plusieurs modalités de mise à disposition des ressources par les fournisseurs IaaS. Nous précisons dans le chapitre 3 les détails de celles-ci. L'utilisation de ressources matérielles du cloud passe le plus souvent par la mise à disposition de machines virtuelles par les fournisseurs, pour le cas d'Infrastructure as a Service [2]. Dans ce cas, le fournisseur de services peut demander l'allocation d'une nouvelle instance, à l'aide d'une interface Web ou à l'aide de scripts spécifiques au fournisseur. Il peut choisir le type d'instance (correspondant au nombre de processeurs, à l'architecture de ces derniers et à la mémoire vive mise à disposition), le type et la taille du stockage. Une fois l'instance créée, le fournisseur de service doit installer [3] et configurer l'ensemble des logiciels nécessaires au bon fonctionnement. Les fournisseurs de cloud proposent habituellement des solutions permettant d'adapter dynamiquement les ressources en fonction de l'utilisation. Elles sont proposées par les principaux fournisseurs tels que les AutoScaling Group d'Amazon Web Services, Azure Autoscale, ou Google Compute Engine. Celles-ci sont habituellement basées sur des métriques telles que l'utilisation du processeur, de la mémoire, etc. Le principe général est l'ajout ou le retrait d'instances en fonction de seuils, le seuil haut montrant une sur-utilisation de l'instance, et le seuil bas une sous-utilisation. Ces solutions restent spécifiques aux fournisseurs, et limitées dans leurs approches. Ce type d'approche, s'il est applicable facilement, fait cependant fi de la structure interne des logiciels et de leur fonctionnement. Or, il semble logique que le flot du logiciel permette de déterminer les besoins du système dans son ensemble. Par exemple, la validation d'un formulaire pourra donner lieu à un ensemble de calculs ou d'envoi de données. Ces calculs prendront de la puissance de calcul, de la bande passante disque pour le stockage des informations, et de la bande passante réseau pour l'envoi et la réception des données du formulaire. Celles-ci sont disponibles en quantité limitées, et différentes selon le type d'instance réservé.

Un autre problème de l'élasticité dans le cloud provient de la durée d'initialisation des instances. En effet, la création d'une nouvelle instance et l'installation des logiciels nécessaires sur celle-ci n'est pas instantanée. Par exemple chez Amazon Web Services, la création d'une nouvelle instance peut mettre jusqu'à 10 minutes selon le type d'instances, à laquelle il faut ajouter le déploiement des logiciels et les reconfigurations et synchronisations nécessaires. Ceci provoque une certaine latence avant l'adaptation des ressources nécessaires. Pour pouvoir garantir une qualité de service définie, le fournisseur de services devra considérer cette non-instantanéité, l'autre solution étant d'avoir des ressources préalablement réservées, ce qui retire l'avantage du prix, celles-ci devant être payées. Pouvoir prévoir les besoins d'une solution logicielle est donc primordial. Une connaissance des moments où un logiciel donné présente ces besoins permet d'adapter le type

2. Infrastructure en tant que service
3. Certaines solutions telles que les AMI d'Amazon Web Services permettent de réaliser cet étape au démarrage de la ressource

et la quantité des ressources louées, de manière à payer un coût adapté pour le fournisseur de services.

Toutefois, dans les logiciels traditionnels, l'ensemble de la logique métier est le plus souvent exprimée à l'aide du code source du programme. Celui-ci n'est pas analysable facilement : il n'est pas aisé d'en déterminer le comportement que ce soit pour les applications centralisées [6] ou distribuées [7]. Les outils BPM ajoutent des informations utilisables : grâce à ces derniers, il est possible de représenter les processus des entreprises sous la forme de schémas exécutables. Ceux-ci permettent de rationaliser et d'obtenir sous une forme utilisable le fonctionnement général des processus des entreprises. En s'appuyant sur les traces d'exécution d'instances de processus et sur leur données, il est possible d'évaluer quantitativement les décisions, et d'évaluer les performances de l'outil en regard de la qualité de service [8].

Nous prenons le parti qu'il est possible d'utiliser les informations structurelles des processus exécutés pour parvenir à l'élasticité de l'exécution des moteurs BPM. L'utilisation et le dénombrement d'éléments internes nécessaires au bon fonctionnement des processus métiers tels que les tâches, les portes logiques, ou les évènements fournit des éléments mesurables, et utilisables dans ce but.

Toutefois, les moteurs d'exécution de tâches BPM sont des applications Web s'appuyant sur des bases de données relationnelles pour la persistance des données, et La mise à l'échelle de ce type de logiciels n'est pas une tâche simple. Des goulots d'étranglement peuvent survenir au niveau du serveur d'applications, du moteur lui-même et plus généralement au niveau de la base de données relationnelle. Ces dernières, le plus souvent utilisées dans les moteurs BPM ne passent pas à l'échelle facilement [9]. Ceci provoque des limites au niveau de la taille d'une installation distribuée. Selon la taille des clients, leur utilisation et leur nombre, il peut ne pas être envisageable d'héberger l'ensemble des clients d'un service sur la même installation. Une distribution des clients sur plusieurs installations, hébergées sur des ressources physiques différentes s'avèrera nécessaire. Nous considérons ce point dans ce manuscrit.

Pour réduire les coûts, les fournisseurs de service hébergent différents clients sur les mêmes ressources. Les architectures multi-tenant permettent d'arriver à ces fins. Celles-ci consistent à mettre sur une même ressource matérielle et logicielle différentes installations de clients différents, de manière isolée et transparente pour ces derniers. Nous proposons dans ce manuscrit des approches visant ce type d'architecture pour les fournisseurs de BPMaaS.

Le passage à l'échelle des installations étant limité, il est nécessaire de pouvoir déplacer les installations d'une ressource à une autre pour pouvoir adapter la capacité aux besoins des clients. Nous nommons ce déplacement *migration*. Par exemple, il faut pouvoir migrer un tenant ayant des besoins excédant la capacité de sa ressource actuelle vers une ressource capable de supporter ses besoins. D'un autre côté, si une ressource est sous-utilisée, il doit être envisageable de déplacer les tenants vers une ressource meilleur marché, afin de réduire les coûts. Pour réduire au maximum les effets sur les durées d'opérations des clients, ces opérations doivent être réalisées alors que le système est en cours d'exécution (migrations à chaud). Nous considérons les migrations à chaud comme une composante importante et les prenons en compte dans nos approches.

Nous présentons dans la section suivante un exemple de motivation, afin de présenter d'une manière simplifiée un problème d'assignation.

2.2 Exemple de motivation

Nous imaginons dans cet exemple un fournisseur de services souhaitant mettre à disposition de clients (que nous nommerons « tenants ») un moteur BPM. Les clients peuvent importer des

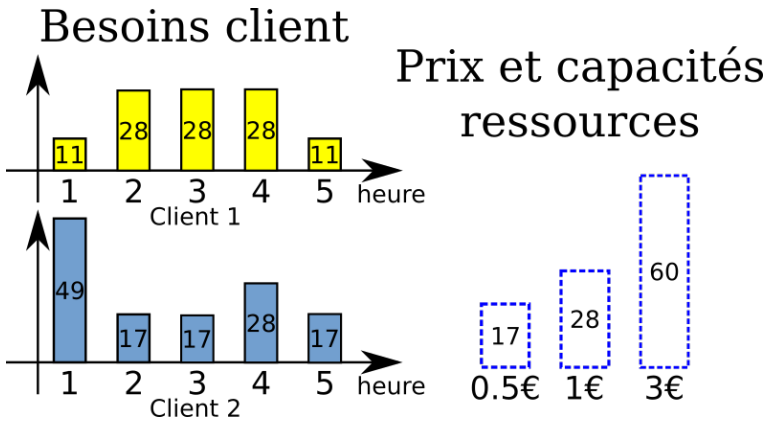

FIGURE 2.2 – Allocation des ressources et attribution

schémas BPM, les exécuter (ces derniers déclenchant des tâches humaines et automatiques) et effectuer des statistiques suite à ces opérations. Dans notre exemple, nous avons deux clients : le client 1 (jaune) et le client 2 (bleu). Pour pouvoir exécuter les processus de ces clients, le fournisseur a des besoins en termes de puissance de calcul, réseau et de stockage. Nous ramenons dans cet exemple l'ensemble de ces besoins sous la forme d'un nombre d'opérations nécessaires pour chaque client, pour un laps de temps donné. Les besoins évoluent dans le temps pour ces deux clients. Nous partons du principe que nous connaissons à l'avance pour chaque laps de temps ces besoins. Dans notre exemple, le fournisseur loue l'ensemble des ressources de calcul sous forme de services IaaS (tels que la location de machines virtuelles, d'espace de stockages) et PaaS (location de bases de données). Ces ressources ont un coût par laps de temps (horaire dans notre cas). Trois types de ressources existent, dont le coût horaire est respectivement de 0.5 €, 1€ et 3€. Ces trois types de ressources ont différentes capacités estimables en termes de nombres d'opérations, respectivement de 17, 28 et 60 opérations. Il est possible de réserver autant de ressources de chaque type qu'il sera nécessaire. Toutefois, le fournisseur souhaite payer un prix minimal pour la location de ses ressources, tout en supportant le nombre d'opérations nécessaire pour chacune des heures concernées.

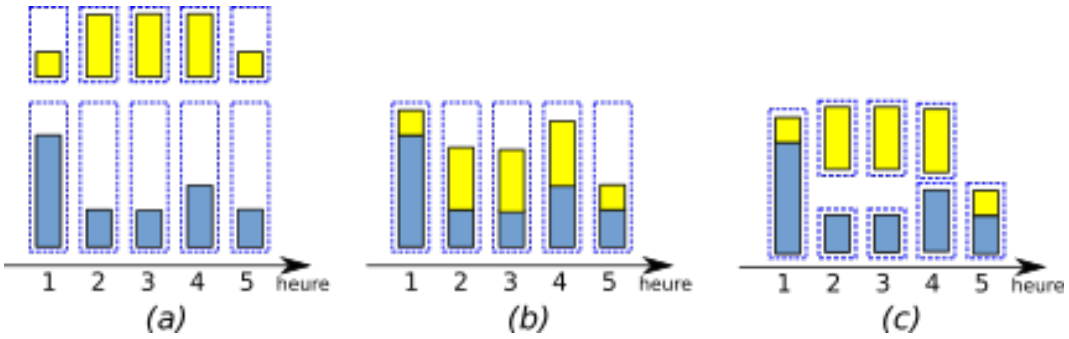

FIGURE 2.3 – Exemples d'allocation de ressources et de distribution de tenants

Nous présentons plusieurs solutions d'allocation de ressources, et d'assignations des tenants sur celle-ci dans la figure 2.3. On peut les décrire ainsi :

— la solution *(a)* présente une solution simple et intuitive où l'on propose pour chaque client une même ressource capable d'héberger ses opérations quelle que soit l'heure concernée. On obtient un coût total de 20 euros pour les 5 heures concernées avec cette distribution.

Cette solution, si elle est simple à déterminer, a pour inconvénient d'être la plus coûteuse.
— la solution *(b)* présente une autre solution similaire où les deux clients sont hébergés sur une ressource nécessaire et suffisante pour chaque pas de temps. Dans ce cas, la distribution coûte 15 euros pour les 6 heures.
— la solution *(c)* présente la solution optimale pour ce problème. A l'inverse des deux premières approches, le nombre et le type des ressources employés est variable : les clients peuvent être distribués sur la même ressource ou sur deux différentes. Le coût total de la distribution représente 9 euros dans ce cas.

Dans les solutions a et b, les clients du service restent hébergés sur la même ressource. La solution c, de coût plus faible, nécessite de « déplacer »les installations des clients jaune et bleu d'une ressource à une autre, à deux reprises. Nous nommons ce déplacement « migration ». Cette migration, qui nécessite un arrêt du traitement du tenant sur la ressource d'origine, une copie de données d'une ressource à une autre, et une activation du tenant sur la ressource de destination aura des effets négatifs sur les performances du logiciel du point de vue du client. Elle peut provoquer des lenteurs dans l'exécution des traitements, ainsi qu'une indisponibilité temporaire du service, pendant une durée significative [10].

Au vu de ces faits, une évaluation des effets des migrations sur les comportements des clients migrés et de ceux hébergés sur les mêmes ressources a son importance. Comprendre leurs impacts est important pour l'élasticité. Nous évaluons dans cette thèse les impacts des migrations en termes d'interruption de service et de qualité de service sur les clients migrés et « colocalisés »sur les ressources concernées.

Notre hypothèse est que les outils BPM, permettant de profiter d'une représentation simple des processus, peuvent nous aider à nous représenter finement les besoins réels du client, et ainsi, à déterminer les ressources nécessaires pour le bon fonctionnement du système. Le but est de permettre aux fournisseurs de BPMaaS (BPM as a Service) de réaliser des économies dans leurs dépenses de fonctionnement, en allouant les ressources nécessaires pour supporter les demandes de leurs clients.

Notre souhaitons trouver des méthodes d'allocation de ressource et d'assignation des tenants permettant d'obtenir des distributions de ressources les moins coûteuses possibles, tout en respectant les contraintes de qualité de service pour les clients. Pour cela, nous définirons les métriques que nous allons employer. En effet, comme nous le verrons dans les chapitres suivants, d'autres critères, tel que le nombre de migrations peuvent être également pris en compte.

2.3 Contributions

Notre contribution est double : nos approches de détermination de la capacité des ressources et des effets des déplacements des clients d'une ressource à une autre (ou migrations) sur les performances globales présentées dans le chapitre 4, et les méthodes d'allocation de ressources et de distribution des clients présentées dans le chapitre 5.

Le chapitre 4 présente l'ensemble des hypothèses que nous avons pris en compte tels que l'utilisation du nombre de tâches BPM comme métrique de mesure des ressources et des clients du système. Nous présentons ensuite notre approche de mesure des ressources, et les résultats obtenus, présentés dans le workshop CloudBPM de la conférence CloudCom à Luxembourg en décembre 2016 [11]. Nous avons utilisé les résultats obtenus comme référentiel de taille de ressources dans les publications concernant les algorithmes. La troisième partie de ce chapitre présente notre approche d'estimation des migrations à chaud sur les performances des clients. Cette approche a été présentée à la conférence Coopis 2018 à Malte en octobre 2018 [12]. Nous

proposons également une approche de migration à chaud pour applications transactionnelles multi-tenant peu invasive, basée sur les tables externes que nous détaillerons dans une future publication.

Le chapitre 5 présente les méthodes que nous avons proposées. La première approche s'applique à optimiser le coût et le nombre de migrations dans une attribution d'un pas de temps au suivant. Elle a été présentée à la conférence IEEE Cloud en juillet 2016 [13]. La seconde partie décrit notre deuxième approche, généralisant la première à un ensemble arbitraire de pas de temps. Nous avons privilégié dans ce cas une approche mono-objectif et proposé un modèle d'optimisation linéaire ainsi qu'une heuristique simple basée sur le concept de segmentation de séries temporelles. Nous avons présenté cet algorithme à la conférence BPM à Barcelone en septembre 2017 [14]. La troisième partie présente notre amélioration de cet algorithme à l'aide d'un algorithme génétique, valable pour notre heuristique restreinte, ou pour toute autre méthode fournissant le même type de résultats, tel que le modèle d'optimisation linéaire correspondant. Cette publication a été publiée à la conférence ICSOC 2018 à Hangzhou en novembre 2018 [15].

2.4 Conclusion

Nous avons présenté dans ce chapitre la problématique de nos travaux, en expliquant le principe d'élasticité dans le cloud, puis illustré avec le problème d'un fournisseur de BPM en tant que service souhaitant fournir ses services à faible coût, tout en soutenant des performances convenables pour ses clients. Nous avons ensuite présenté nos apports sur ce sujet.

Dans le chapitre suivant, nous présentons les différentes notions et concepts nécessaires à la compréhension de nos travaux, ainsi que les approches actuelles répondant à cette problématique.

Chapitre 3

État de l'art

La généralisation du modèle de services à la demande du cloud permet en principe à ses utilisateurs de réduire leur facture en profitant des avantages fournis : apparence de ressources infinies accessibles à la demande, élimination des contrats d'engagement longue durée, et capacité à payer à l'utilisation les ressources [16]. Ceci permet aux fournisseurs de pouvoir adapter la capacité en puissance de calcul, stockage et réseau qu'ils sont capables de mettre à disposition des clients.

Toutefois, comme nous l'avons exposé dans la problématique, même en considérant l'ajout de ressources, les performances des applications ne sont pas forcément proportionnelles au nombre d'instances impliquées. La présence de goulots d'étranglement au niveau de la couche persistance des solutions peut contraindre le passage à l'échelle de l'application, résultant en des performances « sous linéaires »[17]. Ceci est particulièrement le cas pour les SaaS s'appuyant sur des moteurs de bases de données relationnelles et transactionnelles. Celles-ci nécessitent des propriétés ACID au niveau de leurs transactions et s'appuient souvent sur un gestionnaire de transaction unique pour les moteurs les plus connus, ce qui compromet le passage à l'échelle [18]. Ceci est également valable pour les outils BPM, qui sont des applications Web transactionnelles. L'élasticité des solutions transactionnelles n'est pas une tâche aisée.

Nous présentons dans ce chapitre une courte introduction aux concepts en présence : BPM, et cloud, et élasticité. Nous poursuivons succinctement en précisant des informations ayant trait à l'élasticité dans les centres de données, sujet sur lequel de nombreux travaux ont été réalisés. La couche persistance étant un goulot d'étranglement pour les solutions transactionnelles, nous poursuivons avec quelques informations sur l'élasticité des bases de données. L'approche multi-tenant et la prise en compte de migrations sont des critères très importants pour l'élasticité de la couche persistance. Nous fondons notre analyse de l'état de l'art sur ces ceux critères, ainsi que sur ceux considérés généralement dans l'état de l'art des solutions SaaS, que nous décrivons ensuite. Nous terminons par une revue de l'élasticité des outils BPM.

3.1 Business Process Management (BPM)

3.1.1 Définitions

Selon la norme ISO 9000 :2015 [19], un *processus* d'entreprise est un « ensemble d'activités corrélées ou en interaction qui utilise des éléments d'entrée pour produire un résultat escompté ». Une *activité* est « la plus petite tâche attendue dans un projet ». Un projet est « un processus unique qui consiste en un ensemble d'activité coordonnées et maîtrisées comportant des dates

de début et de fin, entrepris dans le but d'atteindre un objectif conforme à des exigences spécifiques, incluant les contraintes de délais, de coûts et de ressources ». Les processus des entreprises sont encore souvent décrits de manière orale ou écrite, mais sans harmonisation. Or, leurs différentes tâches constituantes peuvent être découpées en un ensemble d'activités ayant pour but de contribuer aux résultats d'une entreprises. On appelle ces activités et leurs interactions des processus métiers. Des acteurs ou classes d'acteurs peuvent être associés à ces différentes tâches. Un processus métier peut être par exemple une entrée en stock d'un produit, composée de tâches faisant intervenir plusieurs acteurs comme un gestionnaire, un fournisseur, un responsable. Des règles métiers spécifiques peuvent être déclenchées selon les conditions. Par exemple des conditions telles que "une validation de la part du responsable est nécessaire en cas de commande supérieure à 10000 euros" pourront être représenté par des alternatives dans le flot d'activités. Dans ce cas précis, une activité correspondant à l'intervention du responsable sera nécessaire en plus des différentes étapes.

Une informatisation de la définition des processus et de leurs traitements peut apporter beaucoup d'avantages en permettant d'automatiser et de rationaliser le traitement de ces processus. Plusieurs approches d'informatisation des processus métiers ont été développées comme BPEL, BPM, etc. La plus connue et utilisée est actuellement BPM faisant intervenir BPMN 2.0 [20], standard proposé par l'Object Management Group (OMG). Le principe général est la mise au point de schéma de processus, composés de tâches, branchements, évènements. Un exemple de schéma BPMN est présenté en figure 3.1. Nous présentons chacun de ces éléments dans la sous-section suivante.

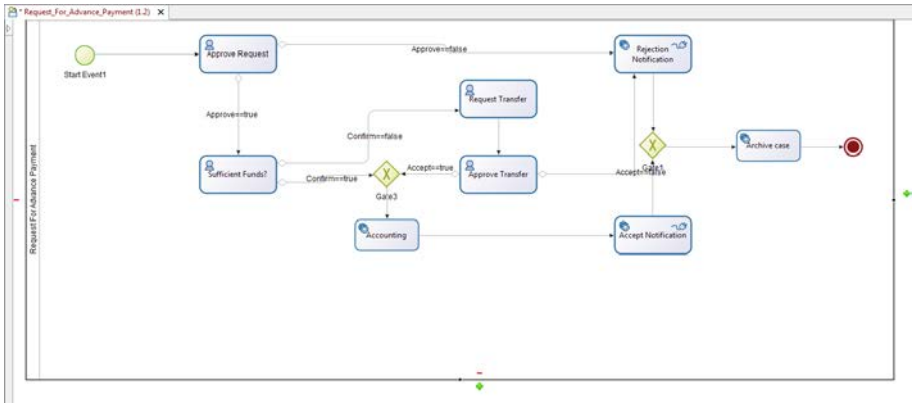

FIGURE 3.1 – Exemple de diagramme BPMN

3.1.2 Éléments de base

Un processus BPMN peut être composé de plusieurs éléments [1]. Celui-ci inclut des objets, des flots de séquences, ou des flots de messages. Les flots de séquences lient deux objets dans un diagramme de processus, et forment une relation de flot de contrôle tel qu'un ordonnancement entre les tâches. Des variables liées au processus métiers peuvent être mises à jour au niveau des différentes étapes. La figure 3.2 présente les éléments composant un processus BPMN. Nous précisons ici les éléments principaux :

— activité : il peut s'agir d'une tâche ou d'un sous-processus. Une tâche est une activité atomique, dont le but est de représenter un travail devant être effectué. Celui-ci peut être automatique, ou humain (représenté alors par un formulaire qui devra être saisi et validé).

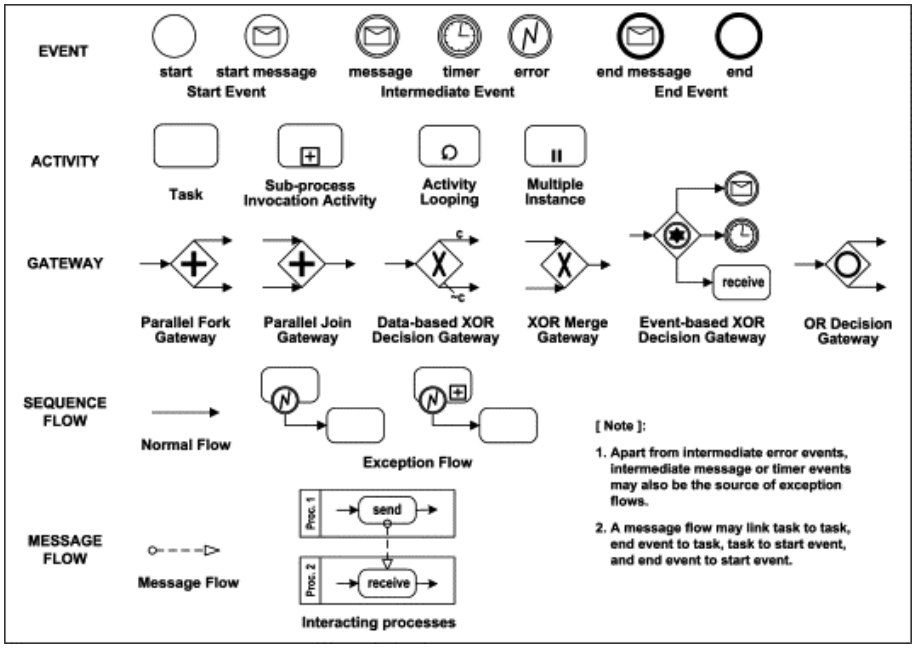

FIGURE 3.2 – Aperçu des éléments BPMN (Dijkman et al. [1])

— branchements (gateways) : il s'agit d'éléments servant à déterminer le routage du flot du processus. Il existe plusieurs types de branchements tels que les branchements parallèles, qui répercutent le flot sur plusieurs transitions de sortie. Les branchements exclusifs et inclusifs sont de leur côté conditionnels, et référencent les variables de processus.
— évènements : un évènement peut signaler le début ou la fin d'un processus, ou peut avoir lieu de manière asynchrone durant l'exécution du processus. Ceux-ci peuvent être déclenchés par différentes situations : un envoi de message, une minuterie, une erreur, etc.

3.1.3 Outils de modélisations et d'exécution des processus métiers

Différents outils aident à la mise au point, l'exécution et le monitoring de processus métier résultant de ces schémas et de leur utilisation. Les plus connus sont des outils de design de schéma, permettant à l'utilisateur de représenter sous forme visuelle les processus de l'entreprise. Les moteurs BPM permettent d'exécuter ces processus et de déclencher les différentes étapes internes. Ces étapes peuvent faire intervenir des systèmes extérieurs (via l'appel de services : mail, Web Services,...), des acteurs du système (via la génération de formulaires). Les flots peuvent être également composés de branchements conditionnels, signaux, communication inter-processus, etc. Des variables embarquées, tant au niveau du processus que des tâches internes sont utilisées pour la persistance des données et les différents branchements. Celles-ci sont initialisées ou modifiées par les tâches et certains évènements. L'ensemble de ces points permet aux outils BPM de former des logiciels dont le comportement est paramétrable. Des utilisateurs formés peuvent ainsi modifier les différents schémas des processus en fonction des évolutions des processus des entreprises.

3.2 Cloud

Le cloud computing consiste en l'exploitation de serveurs distants, habituellement par le biais d'Internet. La puissance de calcul, le stockage, la bande passante réseau, les services et logiciels peuvent être mis à contribution par des fournisseurs (dans le cadre de cloud publics) pour les clients. Ces derniers peuvent être des individus, des entreprises, voire même des états. De nombreux avantages théoriques sont fournis par ce type d'architecture, en particulier grâce aux économies d'échelles liées à la mise en commun des serveurs et à la simplicité d'accès fournie par le Web.

3.2.1 Caractéristiques

Selon la CNIL [21], « *Le cloud computing (en français, « informatique nuagique ») fait référence à l'utilisation de la mémoire et des capacités de calcul des ordinateurs et des serveurs répartis dans le monde entier et liés par un réseau. Les applications et les données ne se trouvent plus sur un ordinateur déterminé mais dans un nuage (cloud) composé de nombreux serveurs distants interconnectés.* »

Selon la définition de l'Institut national des normes et de la technologie (NIST) américain [22], le cloud a les caractéristiques suivantes :

- self-service à la demande : le client a la possibilité de réserver et utiliser du temps de calcul et des outils de manière automatique sans intervention humaine,
- accès généralisé par le réseau : le client a la possibilité d'accéder à ces ressources à travers des mécanismes standardisés ou par le biais d'applications par réseau public ou privé,
- mise en commun des ressources : le client peut accéder aux différentes ressources avec un niveau d'abstraction vis-à-vis de l'infrastructure sous-jacente. Ces dernières sont le plus souvent partagées entre les différents clients (architecture multi-tenant), sans effets directement visibles par l'utilisateur final.
- élasticité rapide : le client peut allouer et libérer ces différentes ressources à l'envie, selon ses besoins. La quantité de ressources disponibles semble infinie à l'utilisateur, même si elle s'appuie en fait sur un nombre de ressources limitées du fournisseur.
- mesure des services : les systèmes des opérateurs permettent un accès à des outils de mesures appropriés concernant les différents services proposés. Les utilisateurs et fournisseurs peuvent ainsi avoir accès au contrôle, ainsi qu'à des rapports concernant l'infrastructure utilisée.

Les ressources mises à disposition peuvent être de plusieurs natures : ressources de calcul, réseau, stockage, mais également mise à disposition de logiciels pour informaticiens (tels que bases de données, frameworks web) ou pour utilisateurs finaux (tels que boites mails, suites de bureautique, etc.). :

3.2.2 Modèles de services

Il existe plusieurs modèles techniques de cloud computing répondant à cette problématique, celle-ci fait intervenir plusieurs niveaux de gestion des couches logicielles par l'opérateur. Dans son analyse taxonomique du cloud computing, Esteves [23] cite les suivants :

- Le modèle *IaaS* (Infrastructure as a Service) consiste en la mise à disposition de ressources de calcul, de réseau, et de stockage pour des clients ayant des compétences en gestion de systèmes informatiques. On peut citer comme exemple les instances EC2 (Elastic Compute Cloud) d'Amazon Web Services ou leurs pendants Microsoft Azure ou Google Compute

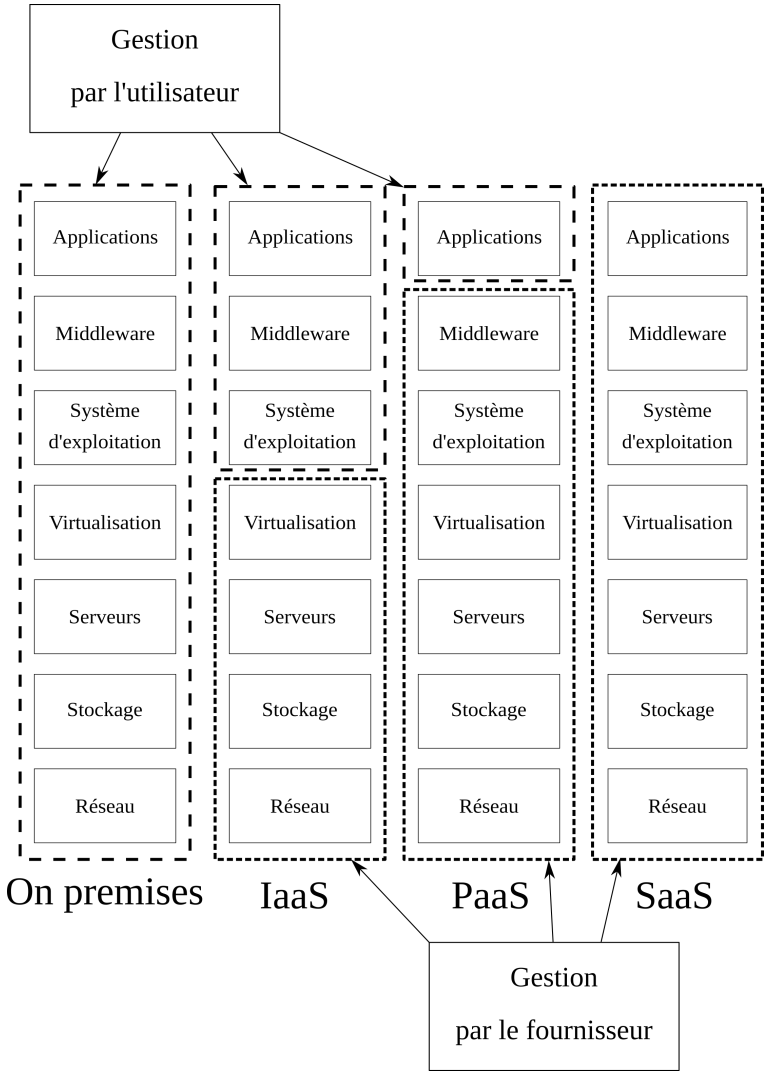

FIGURE 3.3 – Comparatif des architectures cloud standard

Engine. Le client peut choisir parmi une série de types de serveurs de différentes puissances et prix, et louer durant une durée prédéterminée ou à la demande les ressources. Concernant le stockage, il est possible d'allouer des espaces disques et de les relier aux différentes machines virtuelles. Pour la partie calcul, le principe général des fournisseurs est de s'appuyer sur des gestionnaires de machines virtuelles tels VmWare, KVM, QEmu. Ceci permet aux clients d'accéder à des instances isolées.

— Le modèle *PaaS* (Platform as a Service) s'applique à fournir à des clients l'accès à du middleware ou des logiciels. Le client peut alors utiliser dans ses modules logiciels les dites ressources. Les fournisseurs assurent la maintenance de la partie infrastructure (soit par le biais d'offres IaaS, soit en utilisant leur propre infrastructure). On peut citer comme exemple de services PaaS, AWS RDS (Relation Database Services) pour les bases de données, Heroku, Google App Engine.

— Le modèle *SaaS* (Software as a Service) est le plus connu du grand public. Il s'agit ici de la dématérialisation d'applications destinées aux utilisateurs finaux. De manière similaire au modèle PaaS, les fournisseurs peuvent s'appuyer sur les deux couches précédentes ou

utiliser leur propre infrastructure matérielle et logicielle. On peut citer par exemple Google Apps, Office 365, Salesforce.

Le terme « as a Service »(*aaS*) est également utilisé avec la classe de logiciels ou de services utilisés fournies de manière dématérialisée. On peut ainsi voir les formes « Database as a Service », « Desktop as a Service », etc..

Les modèles *BPaaS* et *BPMaaS* ont trait à la gestion en tant que service des processus métiers [24]. Le modèle BPaaS consiste en la mise à disposition aux utilisateurs d'un moteur d'exécution de processus métiers définis et mis à disposition par le fournisseur. Le modèle BPMaaS y ajoute leur modélisation, ainsi que leur gestion à travers les services du cloud. Il s'agit pour les deux de Software as a Service [4] : l'outil mis à disposition consiste en une interface permettant d'exécuter des processus, et de distribuer et valider les tâches correspondantes, auxquelles s'ajoutent une interface de gestion. Avec cette architecture, le client peut ainsi définir ses propres processus métier, et profiter du système sans se préoccuper de la gestion de l'infrastructure. Concernant le marché de BPaaS et BPMaaS, on peut retrouver comme fournisseurs IBM Business Process Manager, ProcessMaker, KissFlow, RunMyProcesses, etc.

Nous présentons dans la section suivante une introduction à l'élasticité dans le cloud, à laquelle nous ajoutons des éléments sur les systèmes multi-tenants, au niveau IaaS, PaaS et SaaS.

3.3 Élasticité dans le cloud

L'élasticité est le degré avec lequel un système est capable de s'adapter aux changements de charge de travail, en allouant et libérant des ressources de manière à ce que à chaque instant les ressources disponibles correspondent à la demande en cours de la manière la plus précise possible [25].

Une des problématiques majeures du cloud computing est l'estimation réelle de la qualité de service des différents services fournis. Selon Kouki et al. [26], les termes généralement employés sont les suivants :

Definition 1. *La qualité de service (QoS - Quality of Service) est la capacité d'un service à répondre aux exigences de ses utilisateurs en termes de disponibilité, de performance, de fiabilité, et de coût.*

La QoS d'un service doit être définie entre les clients et les fournisseurs d'un service. Ceci est fait habituellement à l'aide d'un contrat de niveau de service.

Definition 2. *Le contrat de niveau de service (SLA - Service Level Agreement) décrit la qualité de service attendue par un client envers le service fourni par un fournisseur. Un SLA est composé d'objectifs de niveaux de services (SLO - Service Level Objective), des services et acteurs concernés, ainsi que d'une période de validité et de pénalités encourues par le fournisseur en cas de non-respect de ce contrat.*

Pour pouvoir respecter la qualité de service des clients quelle que soit la charge, les logiciels supportant les services mis à disposition par le fournisseur doivent être capables de passer à l'échelle. Ceci est rendu possible par les ressources mises à disposition du cloud : quand un fournisseur ajoute des ressources, ses services doivent être capables d'en prendre parti et de voir ses performances adaptées. On parle ici de scalabilité des outils. Lehrig et al. [27] définissent ce concept de la manière suivante :

4. Ou de PaaS si le client final n'est pas celui louant ces services

Definition 3. *La scalabilité est la capacité d'une couche de service d'augmenter sa capacité en étendant sa quantité de ressources consommées*

Toutefois, cette capacité ne suffit pas pour un fournisseur. Les ressources utilisées sur le cloud pour héberger les services ont un coût, comme nous avons pu le voir dans la section précédente. Il est important que ce coût soit minimisé, tout en respectant le contrat. A la possibilité d'ajouter des ressources matérielles ou de les étendre en capacité, on doit ajouter la possibilités de réduire leur capacité ou de diminuer le nombre de ressources employées. Ceci permet une gestion *élastique* des ressources. Nous retiendrons la définition suivante :

Definition 4. *L'élasticité est le degré avec lequel une couche de service peut adapter sa capacité à la charge de travail de manière autonome au gré du temps.*

Le passage à l'échelle peut s'effectuer de plusieurs manières, selon les besoins [28]. On parle de :

— *scale-in* et *scale-out* respectivement pour l'ajout de nouvelles ressources matérielles ou le retrait de ces dernières. Le premier cas sera utilisé en cas d'augmentation des besoins des clients comparés aux SLO. Le deuxième, en cas de réduction des besoins, si les SLO sont atteintes. Dans cas, on parle d'*élasticité horizontale*.

— *scale-up* et *scale-down* respectivement pour l'augmentation des capacités des ressources existantes et la diminution de celles-ci. Le coût de ressources plus puissantes étant habituellement plus élevé, il peut être intéressant d'augmenter ou de réduire celles-ci. Dans ce cas, on parle d'*élasticité verticale*.

Des méthodes simples d'élasticité sont proposées en standard par les fournisseurs principaux de cloud. Celles-ci sont basées sur les opérations de *scale-in* et *scale-out*. Nous présentons dans la sous-section suivante ces approches et leurs inconvénients.

3.3.1 Méthodes d'élasticité standard

Les méthodes généralement rencontrées sont dites réactives. Celles-ci sont généralement basées sur la boucle MAPE (Monitor, Analyze, Plan, Execute) [29]. Celle-ci s'appuie sur les étapes suivantes, lancées itérativement :

— *monitor* : cette étape sous-entend l'utilisation d'outils de monitoring. Des métriques liées à l'utilisation des ressources ou aux performances des services sont récupérées.

— *analyze* : cette étape correspond à l'analyse des métriques obtenues par l'étape *monitor*, et éventuellement à l'utilisation de méthodes de prévision. L'état du système présent ou prédit est calculé à partir des métriques récupérées.

— *plan* : une fois l'état courant (ou prédit) connu, l'outil détermine comment adapter les tailles des ressources pour répondre aux besoins.

— *execute* : le planning étant déterminé, l'outil applique les opérations de passage à l'échelle (création ou suppression de ressources) dans cette phase.

Les principaux fournisseurs de cloud mettent à disposition des fonctionnalités basées sur la boucle MAPE permettant la création et la libération automatique de ressources. Celles-ci sont le plus souvent basées sur un système de seuils, tels que les Auto Scaling Group [5] d'Amazon Web Services, Azure Autoscale [6] ou Autoscaler de Google Compute Engine [7]. Ce type de méthode est la plus utilisée, et celle considérée comme la référence en la matière [30]. Le principe général

5. `https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html`
6. `https://docs.microsoft.com/en-us/azure/architecture/best-practices/auto-scaling`
7. `https://cloud.google.com/compute/docs/autoscaler/`

est qu'une ressource supplémentaire est ajoutée en cas de dépassement d'un seuil au niveau de l'utilisation du processeur, de la mémoire ou d'autres métriques habituellement reliées au répartiteur de charge mis à disposition par le fournisseur. De manière similaire, si un seuil minimal (tel qu'une utilisation de moins de 5 % du processeur par exemple) est atteint une ressource peut être dans ce cas retirée. Ces seuils peuvent être absolus ou relatifs selon les fournisseurs. Ils peuvent être choisis par l'utilisateur, ainsi que les métriques considérées, ce dans la limite des possibilités du système.

Ce type de méthode donne de bons résultats mais reste perfectible sur plusieurs points.

— Les pics de charges ne sont pas bien gérés. Selon la granularité temporelle utilisée dans la méthode (période de rafraîchissement, délai avant de reconsidérer une opération d'élasticité) les pics de charge (ou à contrario les baisses) peuvent provoquer un sous- dimensionnement (respectivement un surdimensionnement) pouvant poser des problèmes de qualité de service ou des coûts plus élevés.

— Les solutions commerciales ne considèrent pas le flot de contrôle des applications en se concentrant principalement sur des ressources systèmes. La présence de traitements intensifs peut être déduite de la structure du logiciel. Cette dernière n'est pas considérée dans ce type d'approches. On peut toutefois relever l'utilisation possible de queues de messages dans les Auto Scaling Groups [8] ou de nombre de requêtes utilisées dans les répartiteurs de charges (tels que Amazon Elastic Load Balancer).

— L'utilisation de ce type de solution ne garantit pas le respect de la qualité de service. L'utilisation de métriques systèmes telles que le pourcentage d'utilisation du processeur ne va pas garantir le temps de réponse ou le débit par exemple. Amazon Web Services et Google Compute Engine proposent des métriques liées aux temps de réponse des requêtes pouvant y être rattachées, toutefois celles-ci nécessitent de s'appuyer sur le répartiteur de charges du fournisseur. De par leur nature, elles ne garantissent pas à tout moment le respect des SLO. De plus, l'ensemble des seuils reste à déterminer par le fournisseur. Ceci n'est pas forcément une tâche aisée au vu du nombre d'éléments à régler (seuils minimum et maximum, temporisation), et des SLA du client final.

— Même si ceci est à relativiser au vu de certaines fonctionnalités de planification présentes dans les Auto Scaling Groups d'AWS par exemple, la nature réactive de ce type de méthode ne garantit pas une élasticité immédiate. L'instanciation de nouvelles ressources du cloud est une opération ayant une durée significative, pouvant prendre plusieurs minutes, auxquelles il faut habituellement ajouter l'installation des logiciels, la configuration et le démarrage de ces derniers. Ce délai -qui n'est pas compressible- provoque une latence dans les opérations de scale-out.

— Ce type de solution ne considère pas plusieurs types de ressources. Le principe est l'ajout ou le retrait de ressources identiques afin d'assurer le respect des métriques envisagées. Toutefois, les fournisseurs IaaS proposent habituellement leurs ressources de calcul selon plusieurs capacités (pouvant faire intervenir nombre de CPU, mémoire vive, bande passante réseau, etc.) correspondant à plusieurs prix. Il n'est pas possible de profiter de ces derniers dans ce cas.

— Ce type de solution ne considère pas les applications dans leur ensemble. Une solution logicielle est composée de plusieurs modules pouvant être déployés sur plusieurs ressources différentes tels que des bases de données, répartiteurs de charge, etc. Les ressources considérées ici peuvent être ajoutées ou retirées. Toutefois, dans un tel système réparti, le goulot d'étranglement ne se situe habituellement pas seulement au niveau d'un seul module.

8. `https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-using-sqs-queue.html`

La sous-section suivante présente succinctement les principaux articles que l'on peut retrouver au sujet de l'élasticité dans l'état de l'art.

3.3.2 Élasticité dans les centres de données

Beaucoup de publications concernent l'élasticité dans les « data centers »(centres de données) comme [31] [32]. Leur but est souvent de minimiser le coût en termes de consommation électrique.

L'élasticité dans les data centers s'appuie souvent sur le concept de *migration à chaud* de machines virtuelles (*VM Live migration*) [33] [34]. Le principe est de déplacer une machine virtuelle (mémoire, et données) d'un hôte à l'autre. Plusieurs approches basées sur ce principe s'appliquent à se rapprocher du placement optimal des logiciels sur les ressources, telles que [35].

Une approche connue s'appliquant à résoudre cette problématique est le Google Reassignment Problem, proposé à la société française de recherche opérationnelle et d'aide à la décision (ROADEF [9]) en 2012 [36]. Dans ce problème, Google souhaite obtenir un algorithme permettant un placement efficace de ses processus dans ses datacenters, en optimisant leur placement sur les ressources physique les hébergeant. De nombreuses équipes effectuant de la recherche opérationnelle ont répondu à cet appel et proposé des solutions, la plupart basées sur des méta-heuristiques telles que la recherche à voisinage variable [37] [38] ou sur de la programmation par contraintes [39]. Le problème d'optimisation, fixé par le concours, souhaite obtenir des résultats en considérant un objectif pondéré de coût d'infrastructure, de coût estimé de migrations de services, de logiciels ou de machines, et enfin d'équilibrage des ressources [10]. Cette approche, mélangeant dans la même fonction objectif l'ensemble de ces valeurs peut être compliquée à régler en regard des SLA (quel poids assigner à l'ensemble des coûts de migrations pour assurer un débit ou un temps de réponse constant ?).

Ces méthodes ne ciblent pas la location de ressources à la demande. Dans ce cas, différentes ressources ayant différents prix sont à considérer. D'autres dimensions peuvent être abordées. Nous décrivons dans la section suivante les axes d'analyse que nous pouvons considérer.

3.3.3 Classification des mécanismes d'élasticité

Plusieurs aspects existent dans les algorithmes d'élasticité, telles que la couche logicielle considérée, l'objectif considéré, etc. Nous nous appuyons dans cet état de l'art sur la classification de Al Dhuraibi et al. [2] sur les mécanismes d'élasticité dans le cloud. Plusieurs caractéristiques telles que le type d'architecture (centralisé / décentralisé), le nombre de fournisseurs, ou la cible de l'élasticité ont permis à Al Dhuraibi et ses collègues de classifier de manière claire les différents travaux existants. Voici une liste de ces dernières, que nous pouvons également retrouver dans la figure 3.4 :

- **configuration** : méthode de réservation des ressources. La configuration peut être *rigide* (ressources préalablement réservées) - ou *configurable* (ressources pouvant être réservées à la demande, en avance, etc.).
- **portée** : couche visée par la méthode d'élasticité. Dans la classification, un découpage entre l'élasticité d'infrastructure (portant sur les machines virtuelles ou les conteneurs), l'élasticité d'applications, ou celle de plateformes est proposé.
- **objectif** : but de l'élasticité. Il peut s'agir d'optimisation de performances, de coût, d'énergie (pour les data centers par exemple), de disponibilité, ou de scalabilité (« increase capacity »).

9. `http://www.roadef.org/societe-francaise-recherche-operationnelle-aide-decision`
10. Pour éviter d'avoir une ressource trop surchargée et faciliter le rééquilibrage dans le futur

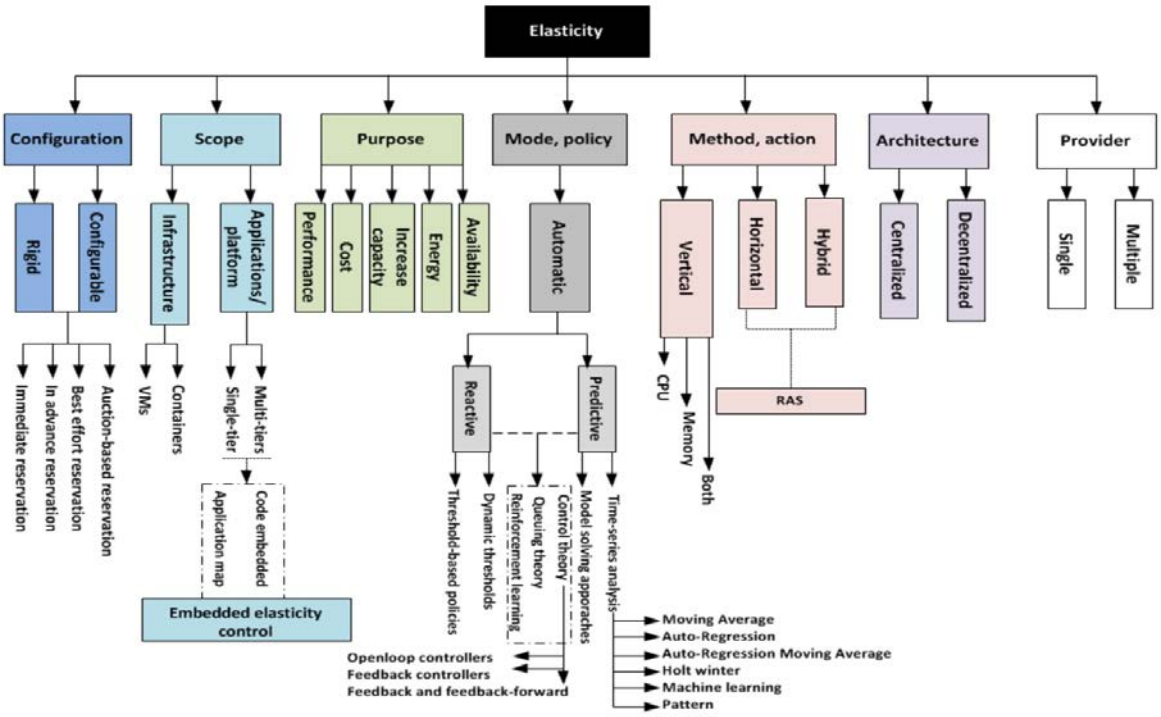

FIGURE 3.4 – Classification des mécanismes d'élasticité (Al Dhuraibi et al. [2])

— **mode** : *réactif* ou *prédictif*. Le mode réactif est basé sur des seuils - dynamiques ou statiques- au-delà desquels des ressources sont créées, modifiées ou détruites. Le mode prédictif s'applique à prédire le niveau de charge futur, et lance des actions en conséquence. Ces approches sont habituellement basées sur l'analyse de séries temporelles ou des mécanismes de résolution d'un modèle simplifiant la réalité. D'autres approches mêlant l'un ou l'autre de ces modes ont été développées, basées sur l'apprentissage par renforcement, la théorie du contrôle ou la théorie des files d'attentes. Nous y ajoutons la catégorie *planifié* qui représente une planification des tâches à réaliser basée sur une connaissance à priori des besoins. Cette étape de planification est nécessaire au mode *prédictif*.
— **méthode** : considération d'élasticité verticale, horizontale ou hybride. Comme nous l'avons vu précédemment, l'élasticité verticale consiste à modifier la taille de ressources actives. L'élasticité horizontale consiste à ajouter ou retirer des ressources. L'élasticité hybride consiste en une combinaison entre les deux méthodes.
— **architecture** : module de gestion de l'élasticité basé sur une architecture centralisée (ressource unique) ou décentralisée (modules distribués sur plusieurs ressources). La première architecture est la plus souvent utilisée dans la littérature, mais certaines approches décentralisées existent dans le but de fournir scalabilité et fiabilité aux modules d'élasticité concernés.
— **fournisseur** : approches compatibles avec l'utilisation d'un fournisseur de cloud (*single*), ou de plusieurs (*multiple*). Cette dernière peut également faire intervenir des clouds hybrides publics ou privés en plus du support de fédérations de fournisseurs de cloud.

Dans cette thèse nous nous plaçons dans le cadre suivant :
— configuration : configurable, basé sur des ressources à la demande,
— portée : élasticité pour moteurs BPM,

— objectif : coût (sous contrainte de performance),
— mode : basée sur une connaissance préalable de la charge nécessaire pour les éléments considérés (pouvant donc être couplées à des méthodes prédictives),
— méthode : élasticité horizontale,
— architecture : nos approches peuvent être utilisées dans les deux cas de figure (centralisée ou décentralisée),
— fournisseur : nous ne considérons pas l'utilisation de plusieurs fournisseurs de cloud différents ou de cloud hybrides.

Nous ajoutons à ce point la considération d'architectures multi-tenant. La multitenancy, approche de partage des ressources du cloud consiste en la mise en commun de manière isolée de ressources de calcul pour plusieurs utilisateurs, ceci permettant la réduction des coûts [40]. Les migrations à chaud, comme nous le verrons dans le chapitre 4.3 sont un élément important dans une architecture multi-tenant pour s'approcher d'une élasticité optimale [41]. Ces migrations, étudiées dans le cadre des machines virtuelles, l'ont également été beaucoup dans le cadre des bases de données en tant que service (DBaaS [11]). Nous les présentons dans la section suivante.

3.3.4 Élasticité des bases de données

Avec l'arrivée du cloud, les bases de données doivent être capables de gérer des débits bien plus élevés qu'auparavant. Toutefois, comme nous l'avons vu précédemment, les bases de données relationnelles passent à l'échelle verticalement, mais plus difficilement horizontalement, à cause des garanties qu'elles offrent. Le problème provient de la gestion des transactions ACID, qui permet une grande sécurité pour les données, mais s'avère complexe à distribuer entre plusieurs machines. Les bases de données NoSQL, de leur côté plus permissives, peuvent voir leurs performances augmenter mécaniquement avec l'ajout de nouvelles ressources. Cette limitation des bases de données relationnelles est en cours d'étude : de nombreuses bases de données *NewSQL* ont vu le jour, tentant de concilier les deux mondes [42]. Toutefois, à notre connaissance, aucune base de données pleinement élastique et naturellement compatible avec les moteurs BPM n'était disponible. D'un côté, des adaptations du code (au niveau du mapping objet-relationnel) rendent l'utilisation de ce type de bases difficile. De plus, certaines fonctionnalités primordiales telles que le support complet des contraintes ACID ou la gestion de transactions distribuées sont rarement disponibles. Ces points sont importants pour certains moteurs BPM, tel que celui de la solution Bonita [12] qui nécessite l'utilisation de requêtes distribuées pour son bon fonctionnement. En effet, la proposition de bases de données NewSQL reste encore très jeune, comme on peut le voir dans le nombre de fournisseurs ou de publications [43] [44] [45].

Les approches multi-tenant peuvent résoudre partiellement ce problème : de nombreux travaux ont été réalisés à ce sujet [10] [46]. La migration à la demande des bases de données relationnelles multi-tenant est une technologie critique pour l'élasticité des applications s'appuyant sur les base de données relationnelles : elle est à classer au même titre que scalabilité, consistance, et tolérance aux pannes [47]. Le principe est de « déplacer »les données de la base d'une ressource matérielle à une autre avec le moins d'effets possibles sur les clients du système.

Il existe certaines bases de données commerciales proposant des solutions permettant les migrations tel que Oracle avec sa solution propriétaire RAC, permettant la migration à chaud de ses instances [48], ou depuis peu, les pools élastiques [13] de base de données de Microsoft Azure qui offrent une approche simplifiée basée sur SQL server via l'utilisation d'unités de transaction

11. DataBase as A Service
12. `https://fr.bonitasoft.com/`
13. `https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-pool`

de base de données élastiques (*eDTU* - elastic Database Transaction Units). Ceux-ci sont basés sur les DTU (Database Transaction Unit), métrique mixant utilisation du CPU, mémoire, débit de transaction, et ratio écriture/lecture. Cette approche est pensée en vue des besoins de multi-tenancy des clients des services de bases de données. Elle permet aux clients de s'abstraire de la gestion des ressources et d'adapter automatiquement l'infrastructure hébergeant les bases de données selon la demande. Des seuils minimum et maximum sont définissables au niveau des pools, afin d'éviter de trop grandes dépenses ou un sous-dimensionnement des ressources. Le niveau d'isolation multi-tenant privilégié est au niveau des instances de bases de données, le client pouvant poser des limites d'élasticité pour chaque base, afin d'éviter qu'un de ses clients consomme trop de ressources et provoque des manques au niveau de ses autres tenants. La possibilité d'effectuer des requêtes multi-base de données est également proposée par ce service.

Des travaux s'appliquant à optimiser le placement des bases de données élastiques ont été réalisés [49] [50] [51] [52]. Schaffner et al. [40] proposent une analogie intéressante avec le placement de jetons de tailles diverses dans un nombre fixe de paniers. Les jetons représentent ici les instances de bases de données, et les paniers les ressources qui vont les héberger. Dans l'exemple fourni, les auteurs parlent d'équilibrage de la taille des jetons entre l'ensemble des paniers. Ils citent comme évolution la minimisation du nombre de migrations.

Nous avons pu voir dans cette section l'importance des migrations à chaud et des architectures multi-tenant pour la couche persistance. Nous considérons ces fonctionnalités dans cette thèse. Dans la section suivante, nous présentons les méthodes multi-tenant d'élasticité pour solutions SaaS.

3.3.5 Élasticité SaaS multi-tenant

Comme nous venons de le voir, les bases de données relationnelles ne passent pas à l'échelle facilement, ou dans des versions spécifiques nécessitant des adaptations au niveau des logiciels. Or, notre considérons dans notre approche l'ensemble des modules nécessaires au bon fonctionnement du moteur BPM, dont les bases de données relationnelles font partie. Nous partons du principe que les migrations étant un point central pour les DBaaS multi-tenant, celles-ci le seront aussi pour les BPMaaS multi-tenant. Nous présentons ici les principaux travaux ayant trait à ce concept.

Saez et al.[53] proposent le framework SCAR, dont le but est d'aider les développeurs à distribuer les composants logiciels sur plusieurs offres du cloud dans le but d'atteindre les objectifs business et les demandes variables en performances des clients. Ce framework, basé sur le concept d'utilité (au sens économique du terme), propose un modèle topologique pour les applications du cloud. Celui-ci comprend les modules d'une application, ainsi que les liaisons entre ceux-ci. Les auteurs proposent également un modèle d'optimisation permettant de prendre les décisions attendues de passage à l'échelle pour les applications. Le modèle proposé est basé sur une vue utilitariste, s'appliquant à optimiser le revenu obtenu multiplié par la satisfaction du client, auquel est retiré le coût, ce durant une période temporelle. La satisfaction est calculée en fonction du gain et de la perte d'utilisateurs, le revenu en termes de gain engendré par l'utilisateur compensé par le taux de disponibilité prévu et la quantité de requêtes émises et en échec. Le coût considéré comprend celui des ressources et des adaptations de l'infrastructure (migrations).

Verma et al.[54] proposent un framework de prédiction et d'allocation de ressources pour systèmes multi-tenants. Celui-ci s'appuie sur un modèle prédictif de court terme et de long terme basé sur l'analyse de série temporelles et des modèles auto régressifs. Les métriques considérées sont le nombre de CPU, leur utilisation, le nombre de fonctionnalités et leur priorité, la période d'activation d'un tenant (actif ou en pause). Le processus d'allocation de ressources est basé

sur un système de seuils et ainsi l'ajout ou le retrait de machines virtuelles pouvant supporter la charge nécessaire pour les tenants, selon leurs états et leurs attentes prédites au moment concerné.

Yang et al.[55] proposent une approche de placement de tenants. Le but est de maximiser le nombre de tenants sans violer leur SLA. Leur approche est basée sur l'utilisation d'algorithmes génétiques, de *raisonnement par cas*, et d'heuristiques, ainsi que d'un modèle d'estimation de ressources. Dans ce dernier, chaque tenant consomme des ressources de calcul, d'entrée/sortie de disque, et de mémoire, et chaque ressource en met à disposition. Le placement des tenants en termes de plans d'exécutions est calculé à l'aide d'un algorithme génétique, couplé à un module de raisonnement par cas qui permet de faire la correspondance entre plans d'exécutions et ressources disponibles. Une approche heuristique basée sur la distance est ensuite utilisée pour sélectionner les meilleures allocations.

Zheng et al.[56] proposent un framework qui se concentre sur la recherche du meilleur moment de migration des données des tenants dans une base de données multi-tenants. Le but est de minimiser le coût de migration en observant la charge. Cette approche est online et s'appuie sur le modèle MAPE. Ils utilisent un modèle d'estimation de demande pour déterminer quand migrer, et un modèle de coût pour guider les choix de migrations. Les besoins des clients sont estimés en termes de besoins en capacité de calcul, mémoire et de stockage. Les auteurs indiquent que le modèle de coût optimise les performances tout en tenant compte de la durée d'interruption et du surcoût dû aux migrations. Toutefois, l'heuristique proposée ne semble pas faire apparaître ces concepts.

Fehling et al. [57] proposent un framework pour l'optimisation de la distribution des tenants dans les applications du cloud. Ils s'appuient sur une métaheuristique basée recuit simulé et la méthode Hill Climbing pour déterminer la meilleure distribution de tenants, sur des ressources, caractérisées par leur prix par instance, puissance de calcul ou SLA. L'assignation à des cloud privés est également supportée. Cependant, dans leur expérimentations ils considèrent la quantité de tenants attribuée à différentes ressources correspondant à différents niveaux de SLA et prix.

Pallavi et al.[58] cible les SaaS, en considérant exclusivement les ressources de calcul en partant du principe qu'un ensemble de processeurs est utilisé pour le traitement des requêtes ou des tâches. Des algorithmes de répartition tels le Round Robin pondéré sont ensuite utilisés pour la distribution des tâches dans des files correspondant à chaque processeur. Une base de données multi-tenant est utilisée pour la persistance.

Espadas et al. [59] proposent un modèle d'allocation de ressources pour passer à l'échelle des applications SaaS. Il est basé sur trois approches : une isolation des tenants séparant les contextes (au niveau persistance et authentification), une allocation de machines virtuelles par tenant implémentant des mécanismes permettant de calculer le nombre de VM par tenant selon son poids en nombre d'utilisateurs, et une répartition de charge envoyant les requêtes à chaque tenant concerné. Les auteurs déterminent la taille des machines virtuelles selon la mémoire et la quantité de threads assignés au serveur d'application. Une heuristique basée sur le problème du sac à dos est utilisée pour trouver des solutions viables.

Chen et Li [60] présentent un modèle réactif dont le but est de maximiser le nombre de tenants assignés tout en respectant la capacité des ressources. Ils étendent un modèle de planification en fournissant une heuristique gloutonne réactive. Celle-ci s'appuie sur le calcul de distance entre vecteurs représentant les ressources nécessitées par les tenants, et celles correspondant aux capacités des ressources.

Certaines approches visent des outils spécifiques telles que [61] pour les applications haute performances basées sur les workflows. D'autres approches architecturales ne considérant pas l'allocation de ressources et la planification, telles que [62] pour les micro-services, ou Kangaroo

pour les VM (comme surcouche d'Openstack) [35].

L'ensemble de ces travaux considère l'élasticité des SaaS multi-tenants, toutefois aucun ne considère les effets des migrations dans leur modèle, mis à part [56]. La structure interne du logiciel quant à elle, n'est jamais abordée, et les métriques de mesure des ressources sont pour tous les travaux exprimés en termes de capacité matérielle. Celles-ci n'a pas de lien direct avec les performances du logiciel. Nous présentons dans la section suivante les méthodes d'élasticité s'appliquant aux moteurs BPM.

3.4 Élasticité et BPM

Les méthodes d'élasticité présentées précédemment ciblent des applications traditionnelles, proposées en tant que service. Les outils BPM, avec la composition flexible des services qu'ils permettent aux utilisateurs, peuvent permettre de raffiner les critères d'élasticité habituellement utilisés tels que l'utilisation du processeur ou le CPU en y ajoutant des informations liées à la structure et à l'utilisation des processus.

De nombreux travaux à ce propos ont été réalisés. Comme le précisent Schulte et al. dans leur état de l'art [63] de l'élasticité du BPM dans le cloud, plusieurs challenges concernant l'infrastructure des outils BPM élastiques existent :

— la planification des processus élastiques. Les tâches des processus doivent pouvoir être exécutées sous contraintes (temporelle, de structure, etc.), tout en utilisant des ressources limitées, quelle que soit l'architecture matérielle utilisée (cloud, on-premises, mixte, ...)
— l'allocation des ressources nécessaires. Les moteurs BPM utilisent une quantité de ressources fixe pour l'exécution de processus. Le Cloud Computing permet d'obtenir de nouvelles ressources à la demande, et ainsi théoriquement d'accéder à un passage à l'échelle pour les fournisseurs et les utilisateurs. La limite reste ici le prix de ces différentes ressources.
— le monitoring de processus et la récupération de données. La capacité des systèmes de BPM élastiques à observer les données d'utilisation courantes, dans le but d'adapter les ressources de manière dynamique vis-à-vis des besoins des clients.
— la coordination décentralisée : un outil BPM traditionnel est au départ un composant simple permettant de créer les processus métier, exécuter et distribuer les différentes tâches les constituant et surveiller leur comportements. Un outil centralisé peut poser des problèmes de passage à l'échelle.
— la gestion de l'état des processus : les processus métier, tâches et autres concepts liés au BPM doivent pouvoir être suivies de manière consistante

Nous nous intéressons aux deux premiers points : l'allocation des ressources, et la planification des processus, et observons l'état de l'art sur ce sujet. Nous reprenons la grille d'analyse décrite dans la section précédente à ce propos.

Il faut différencier l'allocation de ressources de calcul, de stockage et de réseau pour les outils BPM avec l'allocation des ressources humaines pour les tâches humaines. De nombreux travaux ont été réalisés à ce dernier sujet, tels que [64] [65] [66]. Ces travaux ne considèrent que l'aspect ressources humaines et services sans considérer l'exploitation du logiciel, la qualité de service correspondante pour les gestionnaires et le coût total. Les publications suivantes considèrent la gestion de l'infrastructure en ciblant les outils BPM.

Sellami et al. [67] proposent une méthode élastique et multi-tenant pour les processus métiers basés sur BPEL dans le cloud. Ils proposent des spécifications pour un module logiciel se plaçant entre des couches SaaS et PaaS, cette dernière représentant l'outil BPM. Leur approche est basée

sur l'adaptation automatique des ressources dans une boucle infinie à l'aide de patrons d'élasticité. Ceux-ci sont basés sur le métier concerné par les processus métiers, à l'aide d'ontologies. Ils permettent de définir les niveaux de charges nécessaires des processus, en comparant les métriques courantes des patrons avec celles des processus courants. Des actions d'élasticité (scale-in ou scale-out) sont déclenchées si un niveau de charge minimal ou maximal est dépassé au niveau d'une des métriques concernées. Mohamed et al. [68] ont travaillé sur une extension du modèle OCCI pour la gestion élastique des processus métier basés sur les services. Le modèle OCCI offre une abstraction standardisée des ressources du cloud, incluant un modèle de représentation des ressources et de leurs dépendances, ainsi qu'une API permettant de les gérer. Des opérations de duplication, routage et consolidation sont proposées, prenant en compte les processus métier sans les modifier. Leur approche, basée sur MAPE et réactive, permet une adaptation en ligne automatique des ressources. Les opérations sont basées sur le monitoring des processus métier et se basent sur des seuils de temps de réponse pour déclencher les opérations.

Hoenisch et al. [69] définissent un modèle d'optimisation pour l'utilisation élastique des processus, nommé Service Instance Placement Problem (*SIPP*). Dans ce dernier, ils cherchent à minimiser le coût de location d'instances auxquels ils ajoutent des pénalités calculées en fonction du dépassement de dates limites pour les processus, ainsi qu'un coefficient de non utilisation des ressources. Celles-ci sont calculées en fonction d'un coût prédéterminé, lié à la durée en unités temporelles. A celles-ci s'ajoutent des contraintes ayant trait à la charge en termes de CPU et de RAM, qui doivent être respectées. Cet algorithme est ensuite utilisé pour plusieurs pas de temps, avec une adaptation automatique des coefficients de pénalités selon les résultats obtenus. Cette approche cible les cloud privés. L'algorithme a été testé sur le framework *ViePEP* [70] avec une granularité temporelle à la minute. Le modèle SIPP a été repris dans [71] et étendu avec la gestion des cloud publics. Les auteurs ajoutent la gestion du coût de transfert des données, pris en compte dans la colocation des instances. Un ratio de coût supplémentaire pour les cloud publics pour privilégier les cloud privés. Une autre approche concernant l'efficacité énergétique [72] du BPM sur le cloud a également été proposée pour le framework *ViePEP*. Des travaux précédents s'appliquent à minimiser le nombre de machines virtuelles pour ce framework d'une manière réactive, en considérant leur charge en termes de puissance de calcul et d'usage mémoire [73].

Euting et al. [74] détaillent un contrôleur de ressources IaaS élastique pour les processus basé sur la logique des ensembles flous. Celui-ci est capable de déclencher des opérations de scale-in et scale-out à partir de la complexité des instances, de la durée moyenne des processus, du différentiel de ces deux points avec la réalité, et du nombre de ressources de calcul actif. Le contrôleur est lancé périodiquement, ceci permettant une adaptation des ressources nécessaires.

Jaenisch et al. [75] cherchent à optimiser la performance des processus métiers exécutés sur une infrastructure virtualisée. Ils proposent un meta-modèle de déploiement visant à optimiser l'exécution des outil BPM à travers l'élasticité horizontale et testent cette approche en comparant une approche traditionnelle basée sur le niveau de charge du processeur à l'aide d'Auto Scaling Groups d'Amazon Web Services, et sur un contrôleur considérant le nombre de tâches, le temps de traitement et le reste de tâches à effectuer des processus courants.

Amziani [76] propose dans sa thèse un modèle formel pour l'élasticité des processus métiers basés sur les services. Il ajoute aux processus des opérations de duplication et de consolidation des ressources tout en assurant le bon fonctionnement des processus - basés sur les réseaux de Petri dans son cas. Il compare ensuite plusieurs stratégies d'élasticité, plus précisément en observant les violations de qualité de services, l'impossibilité de déclencher des services à cause d'interblocages au niveau du déploiement de ces derniers, et l'absence de phases de duplication suivies de consolidation. Les stratégies d'élasticité consistent en le déclenchement de phase de

duplication, de consolidation ou de routage selon la charge complète, ou la charge par ressource, comparé à un scénario sans modification de l'infrastructure. Des expérimentations implémentant ces algorithmes et basées sur des micro-conteneurs permettent de prouver le bien-fondé de son approche. Un enrichissement de cette méthode permettant la considération d'élasticité verticale et la distinction entre les besoins des différentes requêtes est proposé par Jrada et al. [77].

Rekik et al. [78] proposent un modèle d'optimisation quadratique sous contraintes linéaires pour le déploiement de processus métier configurables dans les fédérations de cloud. Ils se basent sur le concept de processus métier « configurable »qui est un processus métier dont le comportement des tâches et portails peuvent être multiples. Ceci est défini lors de la mise au point du processus et, selon les choix autorisés, permet plusieurs comportements alternatifs lors de l'exécution. L'objectif de ce modèle consiste à sélectionner une configuration pour les processus métier exécutés, et simultanément choisir une configuration de déploiement. La minimisation concerne la somme des coûts d'utilisation des ressources, la somme des coûts de communication intra cloud, et inter cloud. Des contraintes ayant trait à la sécurité, la disponibilité, les capacités des ressources (en termes de processeur, mémoire, et réseau). Toujours sur le sujet des processus métier configurables, Halima et al. [79] proposent de leur côté un modèle linéaire dont le but est l'optimisation du coût sous contraintes. Ils ajoutent des contraintes temporelles ou de besoins matériels au niveau des activités, qui sont ensuite répercutés dans le modèle. Ce dernier prend en considération des fournisseurs et modèles de prix multiples (à la demande, spot ou réservés) au niveau des ressources.

Hachicha et al. [80] étendent également ce concept de processus configurables, en ciblant les systèmes BPM multi-tenant. Ils ajoutent deux opérateurs à BPMN associant ressources configurables et activités : un opérateur d'assignation indiquant l'association entre activités et ressources (ressource réseau, de stockage ou de calcul) ou les quantités minimales et maximales de ressources nécessaires, et un opérateur d'élasticité indiquant les besoins de l'activité concernée, ainsi que le type d'élasticité (horizontale, verticale, ou hybride) envisagée. Un dernier opérateur désigne les activités pouvant partager une ressource dans le contexte de processus partagés entre tenants. Les auteurs étendent ces travaux avec en appliquant un algorithme génétique pour optimiser de manière pondérée le coût des ressources, le temps de réponse et la disponibilité en s'appuyant sur les opérateurs [81]. D'autres travaux liés à l'élasticité des processus configurables ont été réalisés, ayant trait à la simulation de systèmes [82], à la vérification formelle des contraintes temporelles [83]. Ceux-ci toutefois ne concernent pas de manière directe l'élasticité de l'infrastructure. Une approche d'enrichissement des processus métiers basée sur Event B a été proposée par [84].

Mastelic et al. [85] ont développé une technique d'allocation de ressources pour fournisseur de processus métier découpé en 3 phases : prédiction de l'exécution des processus, allocation de ressources et estimation du coût. Ils proposent une représentation des processus sous forme d'un ensemble de tâches, de portails, d'évènements ainsi que de données et de flot de contrôle et de données. Les tâches nécessitent ensuite un ensemble de ressources et ont un temps moyen d'exécution, avec des probabilités d'activation pour les différents chemin des portails. Les ressources du cloud peuvent être de type stockage ou calcul, et possèdent chacune une capacité en termes de CPU, bande passante de stockage et bande passante réseau. Les chemins optimistes (avec la plus haute probabilité) et critiques (nécessitant le plus de ressources) permettent de déterminer les besoins probables ou optimaux. L'étape suivante concerne l'allocation de ressources est un exemple de Fixed Job Scheduling (FJS) et est basée sur un approche Best-Fit, à laquelle les auteurs ajoutent une phase de réutilisation de ressources. Le prix correspondant aux différentes capacités requises est calculé dans la troisième phase.

D'autres approches de planification ne ciblent pas le cloud [64], ciblent les workflows acycliques [86] [87] [88] ou la planification de tâches [89].

Nous décrivons dans le tableau 3.1 une synthèse des différentes publications analysées en fonction des critères décrits dans la section précédente. Nous avons retiré le critère de centralisation - celui-ci n'étant pas prioritaire dans notre approche, et ajouté des critères ayant trait aux effets des migrations sur les performances (*Migr*), au support du multi-tenant (*MultiT*), ainsi qu'à l'utilisation de ressources du cloud ayant plusieurs coûts et capacités (*MultiC*).

Date	Aut.	Conf.	Portée	Mode	Méthode	Four.	Migr.	MultiT	MultiC
2013	[73]	conf.	scalabilité	réactif	horizontal	mono	non	non	non
2014	[74]	conf.	perf.	réactif	horizontal	mono	non	non	non
2014	[75]	conf.	perf.	réactif	horizontal	mono	non	non	non
2014	[67]	conf.	perf.	réactif	horizontal	mono	non	oui	non
2015	[76]	conf.	perf.	réactif	horizontal	mono	non	non	non
2015	[85]	conf.	prix	hybride	horizontal	mono	non	non	oui
2015	[71]	mixte [14]	prix	réactif	horizontal	multi	non	non	oui
2015	[68]	conf.	perf.	réactif	horizontal	mono	non	non	non
2016	[69]	conf.	prix	réactif	horizontal	mono	non	non	oui
2016	[78]	conf.	prix	hybride	horizontal	multi	non	non	oui
2017	[77]	conf.	perf.	réactif	hybride	mono	non	non	non
2017	[80]	conf.	disponibilité	réactif	hybride	hybride	non	non	oui
2017	[79]	conf.	prix	planif.	horizontal	multi	non	non	oui
2017	[81]	conf.	c+p+d [15]	planif.	hybride	mono	non	oui	oui

Table 3.1 – Travaux récents concernant l'élasticité des outils BPM

Beaucoup d'approches sont réactives telles que [68] [67] [74], et ne considèrent pas les modifications futures du niveau de charge. Les opérations de migration, si elles sont parfois considérées [68] ne font cependant jamais partie des critères pris en considération dans les modèles d'optimisation des approches proactives. Les ressources sont la plupart du temps évaluées en termes de métriques système, sauf pour certaines méthodes réactives qui considèrent le temps de réponse des processus [68] [77]. L'architecture multi-tenant, primordiale pour tout SaaS, n'est que très rarement considérée. Les travaux basés sur les processus reconfigurables semblent très prometteurs et ciblent parfois les systèmes BPM multi-tenants [67] [81], toutefois ils nécessitent un enrichissement des schémas des processus ceci passant par des modifications des moteurs BPM.

3.5 Conclusion

Nous avons présenté dans ce chapitre l'état de l'art concernant l'élasticité des outils BPM, et SaaS multi-tenant. Nous avons pu constater l'absence de méthodes reprenant nos postulats. De nombreux travaux s'attellent à fournir des outils BPM élastiques. Toutefois ceux-ci ne considèrent pas simultanément la couche persistance, l'architecture multi-tenant ou nécessitent une adaptation du moteur BPM et des schémas. Pour pallier les insuffisances des approches présentées précédemment, nous avons préparé plusieurs algorithmes. Nos approches considèrent l'ensemble de ces points, et peuvent s'adapter à n'importe quel outil BPM sans modifications. Elles permettent à un fournisseur de BPMaaS de réaliser des économies substantielles tout en respectant la qualité de service du client en considérant des hypothèses réalistes. Nous les détaillons dans le

14. prend en compte cloud privés et publics
15. coût, performances, et disponibilité pondérés

chapitre suivant, décrivant les différentes métriques et postulats que nous avons utilisé dans les méthodes.

Chapitre 4

Hypothèses et qualité de service : outils, mesures, et justifications

Notre but est de fournir des méthodes permettant de minimiser le coût opérationnel de l'infrastructure d'un fournisseur de BPMaaS. Pour estimer les capacités des ressources et les besoins des tenants, nous avons choisi d'utiliser des métriques basées sur les informations de schémas BPMN exécutés, contrairement aux approches basées sur les métriques systèmes (CPU, mémoire, disque, réseau). Les métriques basées sur les informations des processus BPMN exécutés ont du sens pour les utilisateurs finaux et les gestionnaires : ils peuvent rapprocher leur utilisation de leurs processus métier effectifs. Un niveau de charge processeur n'a pas de sens métier, une quantité de processus ou de tâches peut être reliée aux traitements exécutés par les utilisateurs. Toutefois, cela requiert d'estimer la capacité des ressources vis à vis de cette métrique. Il est envisageable de le faire à l'aide de simulations. Celle-ci sera considérée avec le prix des ressources dans les étapes suivantes. D'autre part, nous considérons de manière séparée un autre critère ayant des effets sur la qualité de service : les migrations à chaud. Comme nous l'avons précisé dans la problématique, celles-ci peuvent avoir des effets négatifs sur les performances du système et nous souhaitons les évaluer en regard des temps de réponse et de la non-disponibilité des installations durant celles-ci. Au moment de leur réalisation, il n'existait pas de méthode ou d'outil gratuit permettant d'évaluer ces deux métriques de manière simple et répétable sur le cloud. Nous avons préparé des approches novatrices et des outils permettant de les mesurer.

Nous présentons dans ce chapitre les critères de qualité et hypothèses que nous avons sélectionné pour nos méthodes d'élasticité dans la première section. Dans les deux sections suivantes, nous montrons comment nous avons réussi à évaluer la taille des ressources du cloud pour notre contexte, et à justifier la prise en compte des migrations à chaud, critère caractéristique de nos méthodes. Pour y arriver nous avons mis au point deux framework, correspondant à nos approches. Nous présentons également une méthode de migration à chaud pour applications Web multi-tenant. Nous concluons dans la dernière section.

4.1 Hypothèses retenues pour les métriques employées

Notre but est de proposer des méthodes d'allocation de ressources et de planification pour des fournisseurs de BPMaaS. Dans cette optique, un des problèmes est de pouvoir dimensionner les capacités des ressources et les besoins des clients. Pour cela, nous avons besoin de métriques permettant de mesurer ces concepts. Nous présentons ici nos approches.

Rappelons ici que nous considérons dans le cadre de nos travaux le comportement de moteurs

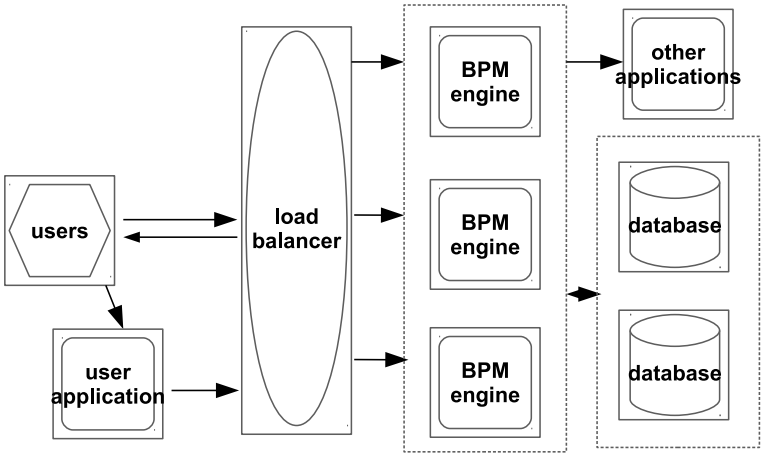

FIGURE 4.1 – Exemple de déploiement en cluster d'un moteur BPM

d'exécution de tâches BPM. Dans notre cas, nous nous concentrons sur des installations « réalistes », comprenant au minimum une ressource de calcul (ou machine virtuelle) pour la partie serveur d'applications, et une machine virtuelle pour la partie persistance. Ces deux ressources auront une capacité de stockage attribuée. Dans le cadre d'installations en cluster (que ce soit pour le passage à l'échelle ou pour la haute-disponibilité), on peut y ajouter plusieurs ressources telles que de multiples ressources pour les serveurs d'applications, ou pour les bases de données. Dans le cadre de serveurs d'applications multiples -chacun contenant une installation valide du moteur BPM- il est d'usage d'avoir un répartiteur de charge afin de router les différentes requêtes. Le moteur BPM peut être utilisé directement via son interface utilisateur, ou à travers des applications tierces. La communication s'effectue alors généralement à l'aide de Web Services. Nous partons du principe que l'hébergement est réalisé dans le cloud, à l'aide de fournisseurs IaaS.

Nous avons effectué dans cette thèse nos expérimentations sur la solution BPM (ou BPMS) Bonita. Bonita [16] est une solution BPM Open Source, initialement développée à l'INRIA [90], et développé par la société Bonitasoft depuis 2009. Bonita propose dans sa solution une architecture séparée en plusieurs points : un studio permettant la mise au point de schémas BPMN ainsi que la génération de formulaires, un portail web mis à disposition des utilisateurs pour gérer les différents processus et tâches, et un moteur capable d'interpréter les fichier BPM du studio, et de démarrer les processus. Un schéma présentant l'architecture est décrit en figure 4.2.

Nos hypothèses sont les suivantes :

— approche multi-tenant
— débit de tâches moteur BPM comme une métrique unique décrivant tenants et ressources
— granularité temporelle
— migrations comme critère

Nous les justifions dans cette section.

4.1.1 Approche multi-tenant

Utiliser les mêmes ressources pour plusieurs clients est une approche naturelle pour des fournisseurs de services. C'est un principe qu'on peut retrouver dans de nombreux secteurs comme dans les transports par exemple. Un fournisseur tel que Fedex n'utilise pas un camion pour chaque client, il groupe tous les colis de ses clients par destination.

16. `https://fr.bonitasoft.com/`

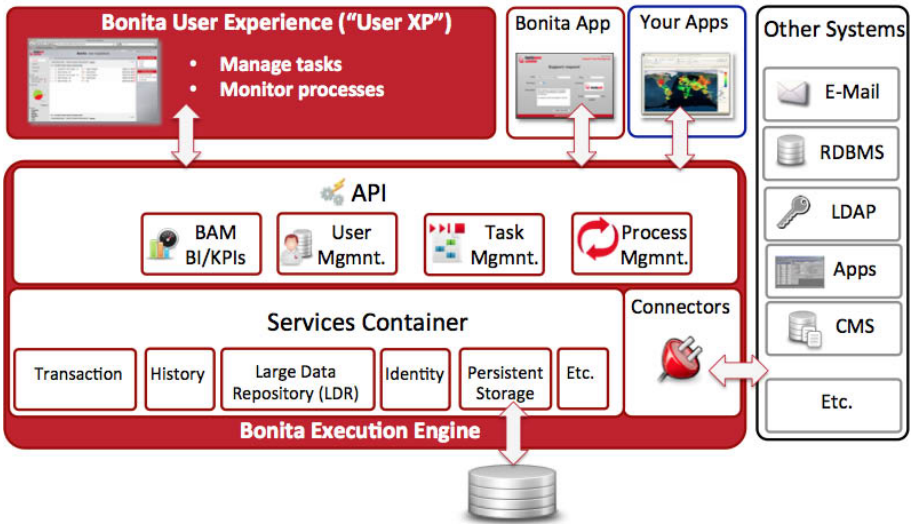

FIGURE 4.2 – Architecture de Bonita

On peut appliquer le même principe pour les architectures SaaS. Héberger les clients de manière isolée sur les mêmes ressources physiques et logicielles apportera une réduction des coûts pour le fournisseur, que celui-ci pourra répercuter ensuite sur les prix de ses services. Un exemple connu est celui de Rightnow cité dans [40]. La mise en commun des ressources au lieu de l'utilisation de machines distinctes pour chaque ressource a permis d'économiser dans leur cas des millions de dollars. On nomme cette approche *multi-tenancy*, où chaque client du service est un *tenant*. Les définitions suivantes, inspirées de celles de Gartner [91] précisent ces différents concepts.

Definition 5. *L'approche multi-tenant (multitenancy) est une référence au mode d'opération où plusieurs instances indépendantes d'une ou plusieurs applications agit dans un environnement partagé.*

Definition 6. *Un tenant est une instance d'un système multi-tenant isolée des autres tenants de manière logique, mais intégrée aux autres de manière physique.*

L'approche d'architecture multi-tenant la plus élémentaire consiste en la séparation complète des installations pour chaque client (serveur d'applications et couche persistance). Cependant, il est possible d'arriver à mettre en commun ces ressources.

Il existe plusieurs moyens de découper les données des tenants. Ceux-ci proposent différents équilibres entre performances globales et isolation. Ce dernier point est extrêmement important, tant au niveau sécurité que performances. En effet, avec une faible isolation des tenants, il existe peu de possibilités d'éviter qu'un tenant consomme l'ensemble des ressources de calcul, de mémoire, de disque ou de réseau du système (problème de famine). Toutefois cette isolation a souvent un coût. On peut dénombrer les architectures suivantes [92] depuis la plus performante (et moins isolée) jusqu'à la plus isolée :

— *shared-table* : le principe est de découper au niveau des tables les tenants, habituellement par le biais d'une colonne spécifiant l'identifiant du tenant. Cette approche permet à la solution multi-tenant de pleinement profiter d'effets de cache entre les tenants, autant au niveau de la couche applicative qu'au niveau de la couche persistance. Le site Salesforce utilise ce type d'architecture.

— *shared-process* : cette approche s'appuie sur l'utilisation de même moteur de bases de données pour plusieurs tenants. L'approche peut être basée sur l'utilisation de tables séparées dans un même schéma, ou sur l'utilisation d'un schéma pour chaque client. Ce dernier point est plus simple à mettre en œuvre vu qu'il n'entraîne pas de modification de la structure de la base selon le tenant concerné.
— *shared-machine* : le principe est de proposer pour chaque client une base de données séparée. Différentes instances du moteur de base de données sont utilisées, et ce pour chaque client. Cette approche est simple à mettre en œuvre, car elle ne nécessite pas d'adaptation du logiciel, tout en permettant d'utiliser cette ressource et le code des applications d'être partagés. Toutefois, chaque instance du moteur ayant accès à sa propre mémoire, il ne peut y avoir d'effet de cache ou d'optimisation au niveau du moteur.

Nous nous concentrons dans le cadre de nos travaux sur l'utilisation du type d'architecture *shared-table*. Dans ce cas de figure, l'application doit gérer de manière interne la multi-tenancy. Nous avons pu voir dans le chapitre 3 que les approches usuelles d'élasticité des outils BPM considèrent la distribution des processus métiers. Dans nos travaux, nous prenons le parti de distribuer les tenants et l'ensemble de leurs données (processus, tâches, utilisateurs, rôles, etc.) dans leur totalité.

4.1.2 Qualité de service des ressources et des tenants

Les moteurs BPM sont habituellement des applications Web, dont la couche persistance est une base de données relationnelle. Comme nous avons pu le voir, cette dernière peut provoquer un goulot d'étranglement : chaque opération réalisée par le moteur BPM est enregistrée et utilise des ressources de calcul, réseau et de stockage. Les performances de la base de données sont donc primordiales pour l'ensemble du système. La métrique choisie pour l'estimation des tailles de ressources et des tenants doit prendre en compte ces postulats : permettre une considération du moteur BPM et de sa couche persistance.

Les moteurs BPM sont des applications de type OLTP (On Line Transaction Processing). Les applications OLTP utilisent habituellement des bases de données avec des charges de travail intensives en écritures. Elles nécessitent le support de transactions ACID [93]. Il existe un critère de qualité habituel pour mesurer les performances d'une base de données relationnelle : évaluer son débit en nombre de transactions par seconde. Une transaction contient habituellement un ensemble d'opérations modifiant les données d'une base de données. Ce fait nous a amené à sélectionner comme mesure le composant le plus élémentaire d'un schéma BPM : l'activité. Comme nous avons pu le voir dans le chapitre 3.1, les activités représentent les tâches atomiques composant les processus. Celles-ci peuvent être automatiques (déclenchant un traitement automatisé) ou humaines (mettant à disposition des utilisateurs un formulaire). L'approche que nous avons choisie est de considérer le débit d'activités (que nous nommerons également tâches) par unité de temps.

Les besoins d'un tenant peuvent être caractérisés par le débit de tâches nécessaires pour l'exécution de l'ensemble de ses processus. Cette métrique est plus précise que le débit de processus, ces derniers étant souvent considéré dans les estimations de performances de moteurs BPM [94]. En effet, les processus peuvent avoir des nombres de tâches différents, ou des flots de contrôle amenant à des exécutions variables, soit à des quantités de transactions variables. D'un autre côté, la modification de l'état d'une tâche nécessite un nombre fixe et peu élevé de transactions. La détermination du débit de tâches peut être fait en observant l'historique d'utilisation, ou en effectuant des calculs liés au nombre de processus, et à leur structure. Il est ainsi possible de déterminer le débit de tâches nécessaire pour un tenant entier. Dans ce manuscrit, nous considérons

l'ensemble des débits en nombre de tâches par seconde.

Concernant les ressources, nous partons du principe qu'il est envisageable de considérer la même métrique de qualité. Pour une configuration donnée, une capacité en termes de débit de tâches peut être fournie par une installation déployée sur un ensemble de ressources matérielles ou de machines virtuelles. Cette capacité est à mesurer en regard du nombre de processus injectés, et du temps de réponse, comme le montre la loi de Little [95]. Nous présentons dans la section 4.2.2 une méthode et un framework permettant de mesurer la taille de ces ressources en termes de débit de tâches. Les résultats que nous avons obtenus nous ont donné des références pour la taille des ressources.

4.1.3 Modèle de coût des fournisseurs IaaS et granularité temporelle

Les modèles de coût des fournisseurs Cloud peuvent s'avérer complexes à évaluer. Comme nous avons pu le voir dans le chapitre 3, plusieurs schémas de facturation existent. Nous nous concentrons sur le plus connu : le modèle de paiement à l'utilisation. Dans celui-ci, les clients paient en fonction du temps d'utilisation des ressources utilisées, habituellement pour l'utilisation de puissance machine : RAM et processeur, ou de la quantité consommée (coût par Gigaoctet pour le stockage, et par Gigaoctet transféré pour les transfert réseau). Ceci est le cas pour les plus connus des fournisseurs : Amazon Web Services, Microsoft Azure, Google Compute Engine.

Nous proposons dans notre approche de discrétiser la dimension temporelle, en tenant compte du modèle de coût des principaux opérateurs. Ceci signifie que pour chaque pas de temps donné, chaque heure par exemple, nous aurons une distribution de ressources et le placement des tenants donnée, mais aussi une mesure des besoins des clients et des capacités des ressources, valable pour l'ensemble du pas de temps. Il nous faut pour cela choisir quelle quantité en débit de nombre de tâches nous allons employer. En effet, les besoins des clients peuvent évoluer durant la durée d'un pas de temps. Pour cela, nous partons du principe que la ressource sur laquelle sera déployée l'installation du client sera capable à tout moment de supporter les besoins du client. Nous prenons donc comme référence la quantité maximale en termes de débit de tâches BPM pouvant survenir durant la durée du pas de temps.

Les distributions de ressources correspondent à la liste des types et coûts pour chaque pas de temps. Celles-ci peuvent être complètement différentes entre un pas de temps et le suivant. Ceci a des effets sur les performances des installations. La figure 2.2 du chapitre 2 représentant deux clients exprime les besoins des clients et les capacités des ressources en termes de débit de tâches.

4.1.4 Les migrations à chaud comme critère

La durée de reconfiguration des ressources est importante : la possibilité d'ajouter des ressources disponibles pour l'utilisateur de manière immédiate rendrait possible une élasticité parfaite. Aussitôt que le besoin de serveurs supplémentaires se ferait sentir, des ressources pourraient être ajoutées, et retirées quand cela n'est plus nécessaire. Toutefois, la durée d'instanciation des ressources IaaS n'est pas immédiate, à laquelle il faut ajouter la durée d'installation, la configuration et le démarrage des modules logiciels nécessaires. Au total, selon les instances employées le fournisseur peut se retrouver à devoir attendre une dizaine de minutes le temps d'avoir une nouvelle ressource disponible. De plus, la redistribution des tenants vis-à-vis des ressources utilisées nécessite une méthode de transfert de l'installation et des données. Dans notre cas, celles-ci nécessitent une copie des données et le démarrage d'une nouvelle installation. Ce type d'opération n'est pas instantané, et a des effets négatifs sur la qualité de service du système pour les

tenants déplacés, et pour les tenants hébergés sur les ressources concernées par le déplacement (origine et destination). Nous verrons dans la section 4.3 une justification de ces points, et une mesure de ces migrations.

4.1.5 Conclusion

Pour récapituler, notre approche a pour but d'assurer l'élasticité des moteur BPM multi-tenant d'architecture shared-table, est basé sur la discrétisation des pas de temps, et utilise le débit de tâches BPM comme métrique de mesure de la qualité de service des clients et des capacités des ressources les hébergeant. Nous y incluons la prise en compte de l'effet des migrations à chaud. Ce principe est généralisable à d'autres types d'applications Web transactionnelles. Nous présentons dans la section suivante notre framework de mesure de la taille des ressources, ainsi que les résultats que nous avons obtenus.

4.2 Estimation de la capacité des ressources du cloud

Nous souhaitons estimer la capacité des ressources pour nos méthodes d'allocation de ressources et de placement de tenants. Pour cela, nous avons besoin d'estimer la puissance des ressources afin de déterminer lesquelles utiliser pour chaque pas de temps, et pour quelle distribution de tenants. Afin d'estimer les capacités des ressources, nous avons préparé un framework capable de réserver des instances du cloud, déployer des installations et de les tester, en regard de notre métrique : le débit de tâches BPM par seconde. Le principe général que nous proposons est de définir et d'exécuter des simulations de charges de travail, pour ensuite mesurer les temps de traitement correspondants. Nous avons besoin de mesures fiables, et reproductibles. Un rapprochement entre les résultats obtenus et les types de ressources du cloud est ensuite réalisé. Nous présentons dans cette section notre approche, et notre framework, ainsi que nos résultats.

4.2.1 Étapes de la démarche proposée d'estimation de la capacité des ressources

Un « bon outil »de benchmarking doit répondre aux caractéristiques suivantes [96] :

— *pertinence* : les résultats de l'outil doivent avoir du sens auprès des utilisateurs. La métrique doit avoir du sens, la méthode employée être représentative du comportement des utilisateurs, les outils utilisés fiable et éprouvés.
— *répétabilité* : l'outil doit permettre de retrouver les mêmes résultats de manière consistante.
— *vérifiabilité* : les utilisateurs du test doivent pouvoir avoir confiance dans les résultats
— *économie* : l'outil doit pouvoir être utilisé de manière économique, c'est à dire de valoir son investissement.

Notre démarche tente d'adresser l'ensemble de ces points.

Comme nous avons pu le voir, les installations de production de moteurs BPM sont souvent composées au minimum d'un serveur d'applications contenant le moteur, et d'une instance de base de données relationnelle. Ceux-ci sont hébergés sur deux ressources différentes. C'est le principe sur lequel nous nous basons pour le déploiement de notre installation.

Vu que nous souhaitons simuler l'interaction d'utilisateurs avec le système, nous devons prévoir un module que l'on nommera « injecteur de processus ». Son but est de réaliser le même type d'opération que ferait un utilisateur : plus précisément dans notre cas, déclencher de nouveaux processus. Celui-ci se représente le plus souvent sous la forme d'un testeur de charge HTTP dans la littérature, comme par exemple l'outil Faban pour le framework Benchflow [97]. Nous

partons du principe qu'un injecteur de processus est déployé sur une ressource différente de celles hébergeant la base de données et le serveur d'application.

Notre but étant d'estimer et de comparer la capacité de plusieurs ressources différentes du cloud, notre framework doit être également capable d'allouer et de libérer des ressources matérielles de type variable, de manière itérative. Une fois l'ensemble des ressources réservées et disponible, il doit être ensuite capable de déployer l'ensemble des modules, exécuter les tests, récupérer les résultats et ensuite de libérer les ressources.

Nous décrivons ci-dessous les étapes d'un test pour un type de ressource donné :

— allouer trois machines virtuelles, une pour la base de données, une pour le moteur BPM et son serveur d'applications, et une pour l'injecteur de processus BPMN ;
— déployer la base de données sur sa ressource ;
— déployer le serveur d'applications, et le moteur BPM sur leur ressource. Le moteur BPM est paramétré pour utiliser la base de données pour sa persistance ;
— déployer l'injecteur de processus sur sa ressource. L'injecteur de processus est paramétré pour injecter des processus dans le moteur BPM ;
— exécuter les tests en lançant l'exécution de processus par l'injecteur selon le niveau de charge défini ;
— récupérer et archiver les résultats obtenus, ainsi que les méta-informations (telles que le nom, le processus employé, les types d'instances employées, etc.) correspondant au test ;
— récupérer d'autres informations importantes (dump de base de données , logs, etc.) ;
— libérer les instances du cloud ;

Un exemple de tests est décrit dans la figure 4.3. On y retrouve un orchestrateur permettant d'initialiser l'ensemble des ressources, ainsi que trois ressources composées chacune de trois machines virtuelles. Les types d'instances pour le moteur BPM et la base de données sont différents dans les trois ressources. Le but de l'orchestrateur est de créer les instances (dans la figure, de type EC2 hébergées sur Amazon Web Services), les configurer, et ensuite les détruire après avoir déclenché les tests et récupéré les résultats. Ces derniers sont archivés avant la destruction des instances (ici sur une espace de stockage S3) pour pouvoir être ensuite analysés.

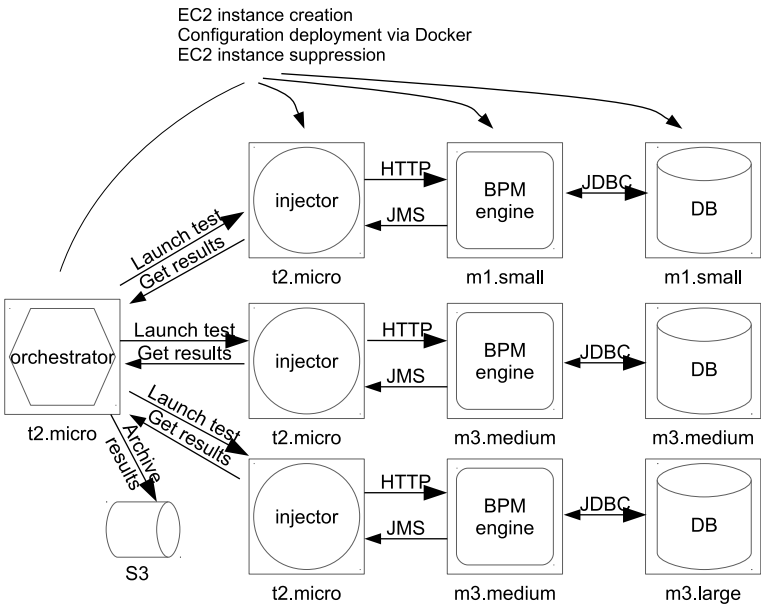

FIGURE 4.3 – Exemple de lancement de tests pour plusieurs configurations

Pour effectuer ces tâches, nous avons besoin d'un outil permettant d'automatiser l'ensemble de ces étapes de manière simple et *répétable*. L'automatisation est une étape nécessaire au vu de la quantité de configurations que nous souhaitons tester. Le but est de potentiellement réutiliser ce framework avec d'autres fournisseurs de cloud, bases de données, moteurs BPM, etc. D'autre part, nous souhaitons que l'outil puisse être personnalisable sur plusieurs paramètres tels que le nombre d'instances de processus testé, le nombre de clients émulé.

4.2.2 Architecture du framework de test développé

Notre framework est basé sur les approches suivantes :

— Utilisation de conteneurs LXC pour la *répétabilité*. Les conteneurs LXC sont une méthode de virtualisation, réalisée au niveau du système d'exploitation (soit une couche plus basse que les machines virtuelles). De nos jours, le logiciel le plus connu permettant leur gestion simplifiée et étendue est l'outil Docker [17].

— Orchestration des scripts. Les différentes étapes doivent être réalisées sur différentes ressources créées dynamiquement, avec des dépendances entre les différents modules. De nombreux outils permettant d'effectuer ces tâches automatiquement existent, tels que Ansible, Puppet, ou Chef.

— Interface utilisateur permettant d'observer le déroulé des expérimentations. Ces dernières peuvent durer plusieurs dizaines de minutes, voire des heures si on considère l'évaluation de plusieurs ressources. L'utilisation de ressources du Cloud nous permet de déclencher de manière parallèle plusieurs expérimentations. Une interface permettant de superviser l'ensemble des points simplifie la mise en œuvre de tests intensifs.

Nous décrivons dans les sous-sections suivantes chacune de ces approches.

Conteneurs : utilisation de Docker

Il existe plusieurs bibliothèques et paramètres dans les applications que nous souhaitons tester. Il peut être difficile de les maîtriser et nous avons besoin d'un moyen de créer des tests isolés et répétables. Pour nous aider sur ce point, nous avons décidé d'utiliser Docker. Docker [18] est un projet open source combinant conteneurs LXC, virtualisation, ainsi qu'un plateforme de gestion de configurations simple. Docker peut être très utile pour l'expérimentation scientifique, en fournissant un moyen simple d'obtenir des configurations logicielles isolées, et ce de manière totalement répétable [98]. Il est également utilisé en entreprise, que ce soit pour des tests ou même pour la gestion de la production. Il s'agit d'un moyen plus simple et plus performant que les solutions habituelles de virtualisation. Docker est basé sur les Dockerfile, fichiers texte contenant la liste des commandes nécessaires pour initialiser un environnement, tels que le système d'exploitation employé, l'installation de bibliothèques et de ressources logicielles, la mise en place de variables d'environnement, et les commandes nécessaires pour l'exécution du programme informatique concerné. Il est également possible de versionner les images Docker, ou de sauvegarder les résultats du lancement de l'ensemble des commandes dans un fichier archive. Ce dernier point peut être important pour retrouver exactement le même environnement d'exécution à chaque fois qu'il est nécessaire.

Dans notre approche, nous nous basons sur une image Docker pour chacun des éléments composant la solution à tester : outil BPM et base de données. Nous utilisons également une image Docker spécifique pour la partie injecteur de processus.

17. `https://www.docker.com/`
18. http ://www.docker.com

Outil d'orchestration : utilisation d'Ansible

Grâce à Docker, nous sommes capables de manipuler des conteneurs en lieu et place d'un ensemble de scripts et de fichiers à configurer et à déployer. Il s'agit d'une étape importante, mais pas suffisante pour nos besoins. Il nous est également nécessaire d'allouer et de supprimer des instances du cloud, et de déployer les différents conteneurs sur ces dernières en paramétrant les conteneurs de la bonne manière. Ainsi, nous devons par exemple garder les différentes adresses IP correspondant à chaque instance créée. Par exemple, le moteur BPM a besoin de l'adresse IP de l'instance hébergeant la base de données, ainsi que les identifiants employés. L'injecteur de processus a besoin de l'adresse IP de l'instance hébergeant le moteur. Celles-ci peuvent être variables, vu que nous souhaitons tester des instances de type (et de prix) différents. Nous avons besoin d'un outil capable d'exécuter des scripts, de réserver, et de supprimer des instances du cloud, de garder l'inventaire de ces instances, ainsi que les différentes variables employées, l'une pouvant parfois être déclarée sur une ressource et utilisée par une autre (comme par exemple l'adresse IP de la base de données). Cet outil devra être capable de fournir un moyen d'effectuer ces tâches. Beaucoup d'outils répondant à ces besoins peuvent être considérés, tels que Puppet, Ansible, Chef, Salt [99]. Nous avons décidé d'utiliser l'outil Ansible. Ansible [19] est une plateforme logicielle développée en Python, dont le but est de gérer la configuration logicielle, la gestion de configurations et le déploiement d'applications. Il est capable de réaliser ces différentes tâches à l'aide de fichiers YAML. Un point important est qu'il ne nécessite pas de composant du côté client (c'est à dire sur les machines ciblées) : il utilise SSH de manière interne pour l'ensemble des communications avec les machines ciblées. Des détails supplémentaires sur Ansible sont disponibles en annexe.

Interface homme-machine : utilisation de Jenkins

Même avec l'orchestration offerte par Ansible, lancer ces scripts manuellement est peu efficace, et peut devenir compliqué au vu du nombre de tests nécessaires. Comme nous le verrons dans la section suivante, nous avons effectué nos tests sur plus de 70 configurations différentes, répété six fois. Il nous était nécessaire de trouver un outil capable de fournir un moyen visuel simple de lancer les tests avec tous leurs paramètres et de consulter leurs avancées. Ceci facilite la *vérifiabilité* du framework. Jenkins [20] est un outil d'intégration continue open-source. Il permet de préparer en premier lieu des compilations de projets logiciels et d'exécuter des tests unitaires à l'aide d'une interface utilisateur Web. Ses capacités ne s'arrêtent pas là : il est également possible de déclencher des scripts shell ou d'autres commandes. L'avantage est que chaque lancement est horodaté et historisé, ainsi que les logs et résultats. Nous l'avons utilisé pour sa simplicité, sa capacité à préparer des scripts, à montrer les lancements en cours et les résultats obtenus. Des détails concernant Jenkins sont disponibles en annexe.

4.2.3 Paramètres de l'expérimentation

Dans cette section nous décrivons les différents paramètres que nous avons employés dans notre expérimentation.

19. http ://ansible.com
20. http ://jenkins.io/

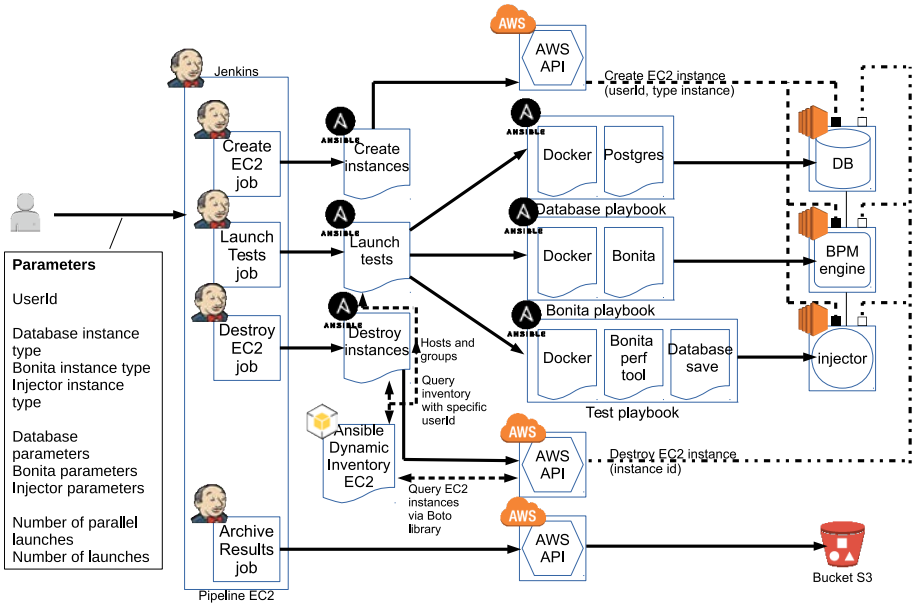

FIGURE 4.4 – Architecture du framework pour des tests basés sur des instances EC2

Aperçu de l'architecture employée

La figure 4.4 présente l'architecture globale que nous avons employée pour nos tests. Nous y montrons les éléments d'orchestration pour chaque partie ainsi qu'un exemple de configuration générée. Une architecture similaire est employée, avec seulement une instance RDS en lieu et place d'une instance EC2 dans notre expérimentation.

Nous avons divisé nos tâches en deux parties : d'un côté des tâches (jobs, voir annexe A) paramétrables correspondant à chaque playbook Ansible (création d'instances, déploiement des outils et lancement de tests, et destruction d'instances), auxquelles nous avons ajouté un job pour l'archivage des résultats. Un pipeline (voir annexe A) paramétrable s'occupe d'orchestrer les différents tests.

Moteur BPM étudié : Bonita

Nous avons utilisé le moteur de la solution Bonita 7.3.1 dans sa version communautaire comme moteur BPM avec Postgresql comme base de données pour la partie persistance. Bonita et Postgresql fournissant tous deux des images Docker, nous avons naturellement utilisé comme conteneurs leurs images.

Injecteur d'instance de processus : outil interne Bonita

Il nous était nécessaire d'avoir un composant permettant d'émuler des interactions de clients. Ceci passe par le déclenchement de processus via les API de Bonita. Ainsi, nous nous sommes appuyés sur un outil interne de test développé à Bonitasoft. Celui-ci permet de déployer des modèles BPMN dans une installation Bonita, de les exécuter en injectant les bons paramètres de processus, de récupérer des métriques globales concernant les performances d'exécution des processus, et enfin de retirer les processus. Les résultats que fournit cet outil sont centrés sur les processus (temps d'exécution, débit, etc.).

Fournisseur cloud : Amazon Web Services

Pour nos tests, nous avons utilisé Amazon Web Services au vu du rôle de leader qu'a ce fournisseur. D'autres fournisseurs existent tels qu'Azure, Google Compute Engine. Le principe reste le même pour ces autres fournisseurs. Nous avons utilisé des instances RDS pour la couche base de données, et des instances EC2 pour les autres couches. Les instances EC2 sont des instances de calcul : des machines virtuelles louées à la demande ou réservées (louées sur une plus longue durée). Les instances RDS sont des bases de données mises à disposition des utilisateurs, basées sur un modèle de coût similaire. Concernant la dimension financière, nous avons considéré le coût horaire de chaque instance, tel qu'il était au moment des tests [21].

Paramètres utilisés

Pour cette expérimentation, nous avons décidé d'utiliser des instances correspondant plus précisément à nos besoins, en l'occurrence des instances RDS de la famille *db.r3* (optimisée au niveau mémoire et stockage) pour la couche persistance, et des instances EC2 de la famille *c4* (optimisée au niveau CPU) pour la couche serveur d'applications et moteur BPM.

Pour nos tests, nous avons utilisé le « processus standard »- processus de référence utilisé à Bonitasoft pour les comparaisons de performances entre les différentes versions de l'outil Bonita. Ce processus est composé de 20 tâches automatiques séquentielles, chacun mettant à jour 15 variables de processus à l'aide d'une constante, et exécutant ensuite un connecteur calculant le nombre de Fibonacci de 25 à l'aide d'une méthode récursive. Cette dernière méthode est lancée jusqu'à ce que 150 millisecondes soient passées. Nous avons exécuté ce processus 3000 fois, pour chaque scénario. Un scénario de test reprend l'ensemble des étapes décrites dans la section 4.2.1.

Nom	Rôle	Usage
name	global	nom du test
perf_tests	benchmark	type du test employé
perf_nb_parallel_launch	benchmark	nombre d'injections de processus en parallèle
perf_nb_launch	benchmark	nombre total de lancements
test_userid	global	identifiant unique du test

TABLE 4.1 – Paramètres principaux du framework d'estimation de la capacité des ressources

Le but de notre expérimentation est de trouver la capacité des différentes ressources du cloud en termes de débit de tâches exécutées par secondes. Une tâche pouvant être extrêmement rapide à lancer, nous avons pris un temps de référence au niveau du processus. Plus précisément, nous avons considéré les débits pour une durée moyenne de 10 secondes du lancement du processus. Au vu de la nature du processus, cette dernière est réaliste. En effet, si on regarde seulement la partie évaluant le nombre de Fibonnacci, on obtient 3 secondes, et ceci sans compter le temps nécessaire pour instancier le processus, allouer les variables, évaluer le workflow, et la fin du processus. Comme nous le verrons dans la section 4.2.4, la durée d'un processus lancé pour un utilisateur seul est d'environ 5 secondes.

Nous avons testé le moteur avec différents nombres d'exécutions de processus en parallèle, représentant différents nombres d'utilisateurs, ainsi que pour différents types d'instances EC2 et RDS pour la base de données et le serveur d'applications. Les composants ont du également être paramétrés en fonction des capacités des instances les hébergeant : le paramétrage du moteur

21. La granularité de facturation est passée à la seconde avec première minute indivisible chez ce fournisseur depuis le 2 octobre 2017.

BPM a été adaptés au niveau du nombre de threads affecté aux exécutions, que nous avons basé sur le nombre d'utilisateurs afin de tirer parti du parallélisme. Le nombre de connexions à la base de données a suivi la même règle. Nous avons également adapté la quantité de connexions à la base de données ainsi que la mémoire affectée à la machine virtuelle Java du serveur d'applications, selon la taille de la machine. Ceci était changé dans les options de lancement de Java au niveau du serveur d'applications. Le paramétrage des instances RDS (bases de données) a également été modifié à l'aide de groupes de paramètres afin d'adapter convenablement la taille de certains buffers en regard des besoins du moteur. Un *groupe de paramètres* est une liste de paramètres pour instances RDS ayant trait à la configuration de la base de données concernée. Ceux-ci sont réutilisables d'une instance à l'autre. Des détails sur les principaux paramètres employés sont décrits dans le tableau 4.1. Ceux-ci sont découpés en 4 rôles, déterminant le module concerné (base de données, moteur BPM, outil de benchmarking, et global). Le rôle global correspond aux paramètres concernant l'ensemble du test, tel que son nom ou son identifiant. Les paramètres *worker_size* et *max_threads* déterminent le nombre de processus légers utilisés respectivement pour l'exécution des tâches par le moteur BPM, et pour les connexions du serveur Web. Pour des raisons de simplicité, nous n'avons pas fait figurer l'ensemble des paramètres techniques nécessaires à l'authentification à la base de données ou au moteur BPM, ainsi que d'autres paramètres liés aux versions employées.

Nom	Couche	Usage
name	global	nom du test
nb_launch	global	nombre de lancement du test
database_instance_type	**base de données**	**configuration de l'instance de base de données**
bonita_instance_type	**moteur BPM**	**configuration de l'instance du moteur BPM**
parallel_launch	benchmark	**nombre de clients en parallèle testé**
configuration	BPM	fichier de configuration Bonita
worker_size	BPM	nombre de threads de travail du moteur BPM
bonita_version	BPM	version du moteur
test_userid	global	identifiant unique de test

TABLE 4.2 – Paramètres globaux du pipeline

Le tableau 4.2 précise l'ensemble des paramètres que nous avons définis au niveau des procédures de lancement. Il s'agit d'une liste simplifiée, comparé aux paramètres nécessaires pour chaque test. Les paramètres que nous avons fait varier dans cette expérimentation figurent en gras. Les autres paramètres sont déductibles de ceux-ci ou sont définis de manière statique, comme par exemple 3000 pour le nombre de lancements du test.

4.2.4 Résultats de l'estimation des tailles des ressources du cloud

La figure 4.5 présente le temps moyen d'exécution des processus comparé au nombre d'injections de processus en parallèle. Les barres d'erreurs représentent les temps d'exécution minimum et maximum obtenus. La ligne rouge représente la référence de 10 secondes utilisée pour la durée d'exécution du processus. Au vu de la longueur et du coût des tests, pour chaque configuration matérielle testée, nous avons expérimenté avec un nombre limité de nombre d'instances de processus injectés en parallèle. On remarque ici le caractère linéaire des résultats au-delà de 50 processus en parallèle pour la configuration *db.r3.4xlarge/c4.8x.large*. Ceci s'explique par le fait qu'en dessous nous ne profitons pas totalement de la capacité de la ressource. Au-delà, le débit

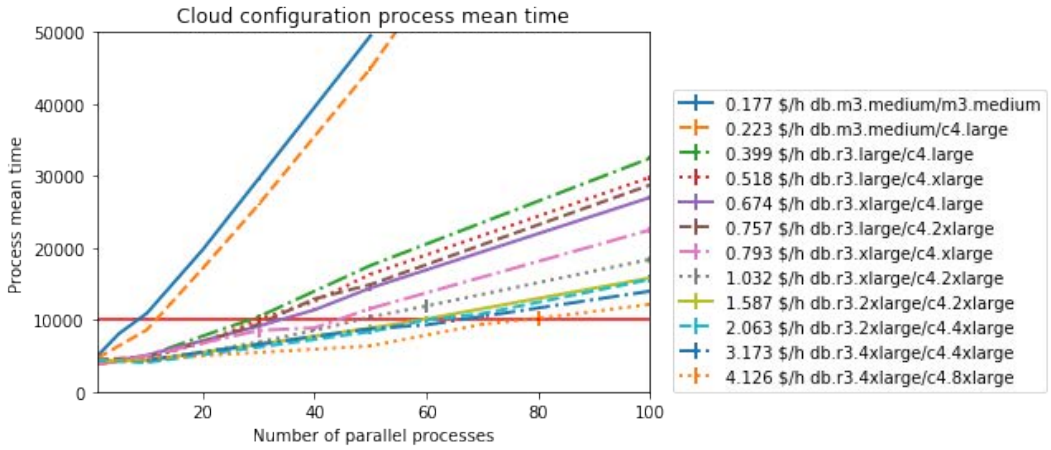

FIGURE 4.5 – Temps moyen d'exécution en fonction du nombre de processus exécuté en parallèle.

maximal de processus est atteint, et ajouter des processus en parallèle ne fait que décaler les traitements.

Notre but étant de calculer un débit en termes de tâches par secondes, nous avons effectué des calculs afin de l'obtenir. Comme nous avons indiqué dans la description de l'expérimentation, nous avons pris pour référence un temps de traitement de processus de 10 secondes. Ceci nous donne le nombre maximal d'utilisateurs pouvant utiliser la solution si on ne souhaite pas dépasser cette durée moyenne. Afin d'obtenir le débit de tâches, nous avons considéré le nombre d'utilisateurs correspondant à cette durée, ce pour chaque configuration. Pour l'obtenir, nous avons extrapolé à partir des résultats existants, en effectuant une régression linéaire entre les deux résultats correspondant à la durée la plus proche de 10 secondes. Ceci nous a donné le nombre d'utilisateurs en parallèle maximal pour cette durée, que nous nommerons p. Nous avons calculé cette valeur pour chaque configuration. Ensuite, pour le calcul du débit de nombre de tâches, nous avons multiplié par le nombre de tâches par processus (soit 20) le nombre de processus exécutés (3000 dans notre cas). Ceci a donné 60000 tâches exécutées, que nous avons divisées par la durée du traitement complet de l'ensemble des processus, en secondes. La figure 4.6 présente ces résultats exprimés en fonction du nombre d'utilisateurs. Les points rouges correspondent aux résultats pour un nombre d'utilisateurs en parallèle égal à p, ce pour chaque configuration. Le débit correspondant en termes de nombre de tâches par secondes représente la valeur que nous avons prise en compte dans nos résultats.

L'ensemble des débits correspondant à chaque ressource est présenté dans le tableau 4.3. Comme on peut s'y attendre, en général un prix plus élevé sous-entend une capacité plus étendue. On peut également remarquer que les débits dépendent des deux types d'instances employés : bases de données et moteur BPM. Pour une même instance de base de données, une instance plus puissante pour le moteur BPM permettra d'obtenir un meilleur débit. Les instances génériques *m3* sont moins puissantes. Un autre point à retenir est que bien que le nombre de processeurs et la quantité de mémoire soient doublés entre chaque type d'instance, le débit n'est pas doublé. Ceci est valable pour les instances *r3* et *c4*, dans l'ordre suivant : *large*, *xlarge*, *2xlarge*, *4xlarge*, *8xlarge*.

Nous y avons ajouté le prix horaire de la ressource ainsi que le débit par dollar (correspondant à la division du débit par le prix). Cette dernière métrique est importante pour déterminer quel type de ressource est la moins chère pour garantir une qualité de service. Nous pouvons constater

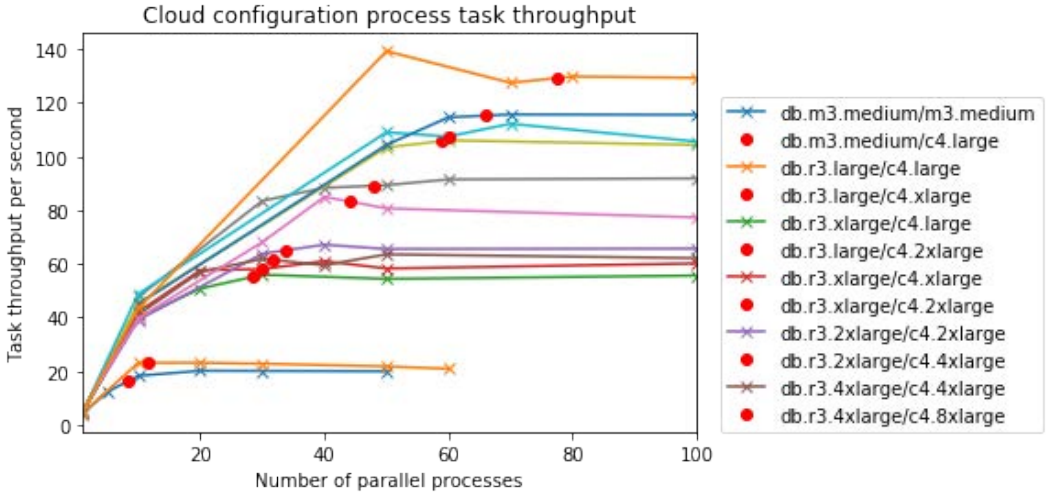

FIGURE 4.6 – Débit obtenu pour chaque nombre de processus simulé en parallèle.

que les ressources les moins chères (et les moins puissantes) sont plus intéressantes, mis à part pour le type d'instance générique *m3.medium* utilisé pour la base de données ou pour la base de données et le moteur BPM. Ceci ne signifie pas que cette configuration est à écarter, car elle reste moins chère dans le cadre de clients nécessitant une capacité moins élevée en termes de débits de tâches. De manière similaire, les instances plus puissantes sont de plus en plus chères à utiliser en regard du débit, mais restent nécessaires en cas de besoins de grande capacité. La configuration *db.r3.large/c4.large* est la moins chère à utiliser (avec le débit par dollar le plus élevé : 138.35 tâches pour chaque dollar investi). Celle-ci est à comparer avec le débit par dollar de la plus grande configuration disponible, 5 fois moins élevé (31.33 tâches par dollar pour une configuration *db.r3.4xlarge/c4.8xlarge*). Pour un fournisseur, la configuration *db.r3.large/c4.large* est clairement à privilégier le plus possible si les besoins des clients sont compatibles.

Un autre point intéressant à noter est qu'augmenter la taille du serveur d'application sans augmenter la taille de la base de données peut devenir contre-productif (et vice versa). Par exemple, la configuration *db.r3.large/c4.2xlarge* est plus chère en termes de débit par dollars que la configuration *db.r3.large/c4.xlarge*. Ceci est valable également pour la configuration *db.r3.large/c4.large*. Un fournisseur de BPMaaS souhaitant optimiser son coût devra prendre bien soin de vérifier quels couples de types de ressources en fonction du prix sont les meilleurs à employer. On peut également retrouver cela dans la figure 4.7, qui présente le prix de chaque ressource en fonction de sa capacité en termes de débit de tâche. Celle-ci montre qu'au-delà d'un dollar par heure, le gain en capacité de la ressource est très faible.

Nous avons présenté dans cette première section notre framework permettant d'effectuer des tests d'exécution de moteur BPM sur différents types d'instances du Cloud [11]. Grâce à l'automatisation fournie par ce framework - basé sur une série d'outils open-source- il nous a été possible d'exécuter ces tests sur de nombreuses configurations aisément. Le coût de lancement a été également réduit, grâce à la libération automatique des ressources en fin de tests. Ce framework peut également être utilisé on-premises (cela a été le cas à Bonitasoft dans le cadre de tests internes), en retirant les phases de réservation et de libération de ressources, et en déclarant un inventaire statique.

Ce framework nous a permis d'obtenir des mesures réalistes concernant la capacité des ressources du Cloud, ainsi que d'avoir des idées générales sur le comportement des ressources du

type inst. base de données	type inst. BPM	prix	débit de tâche	débit par $
db.m3.medium	m3.medium	0.177	16.400	92.656
db.m3.medium	c4.large	0.223	23.157	103.845
db.r3.large	**c4.large**	**0.399**	**55.164**	**138.255**
db.r3.large	c4.xlarge	0.518	58.067	112.100
db.r3.xlarge	c4.large	0.674	65.113	96.607
db.r3.large	c4.2xlarge	0.757	61.474	81.208
db.r3.xlarge	c4.xlarge	0.793	83.236	104.963
db.r3.xlarge	c4.2xlarge	1.032	89.149	86.384
db.r3.2xlarge	c4.2xlarge	1.587	105.794	66.663
db.r3.2xlarge	c4.4xlarge	2.063	107.585	52.150
db.r3.4xlarge	c4.4xlarge	3.173	115.283	36.332
db.r3.4xlarge	c4.8xlarge	4.126	129.279	31.332

TABLE 4.3 – Prix, débit et débit par dollar pour chaque type de ressource étudié. Les formes identiques représentent le même type d'instance de base de données. Les couleurs identiques représentent le même type d'instance de moteur BPM.

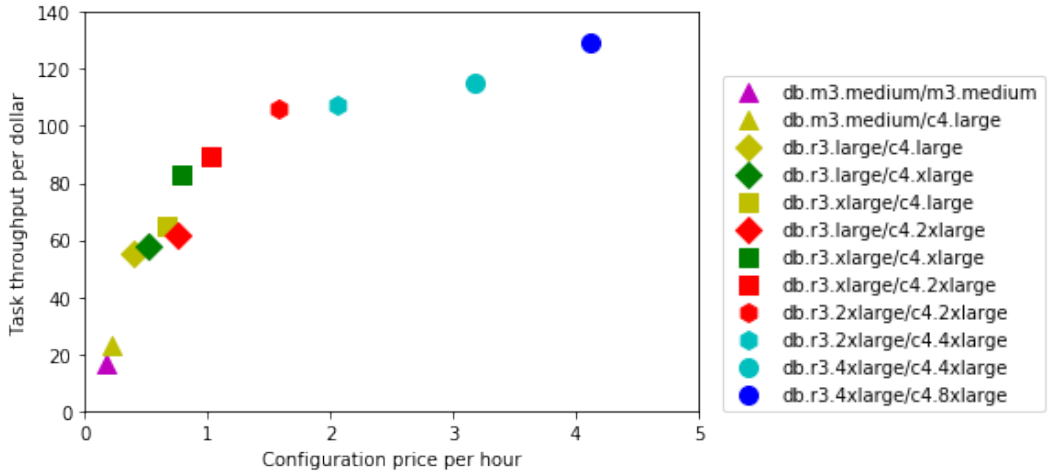

FIGURE 4.7 – Débit, prix, et prix par dollar pour chaque type de ressource

cloud en fonction de leur coût. Les mesures décrites dans cette section ont été utilisées pour l'estimation du coût et la capacité des types de ressources dans nos algorithmes d'allocation de ressource et de distribution de tenants, comme nous le verrons dans le chapitre 5.

Ce framework a également servi de base a l'extension payante Bonita Continuous Delivery [22] (BCD) de la solution BonitaBPM. Celle-ci est dédiée au déploiement de l'outil Bonita sur le cloud, ainsi qu'à la mise en place d'environnement de tests automatiques pour les clients.

4.3 Effets des migrations à chaud sur la qualité de service

La facture des utilisateurs du cloud peut monter très vite. En effet, les besoins des tenants sont souvent variables - il suffit d'imaginer des clients utilisant l'outil seulement aux horaires de

22. `https://documentation.bonitasoft.com/bcd/1.0/`

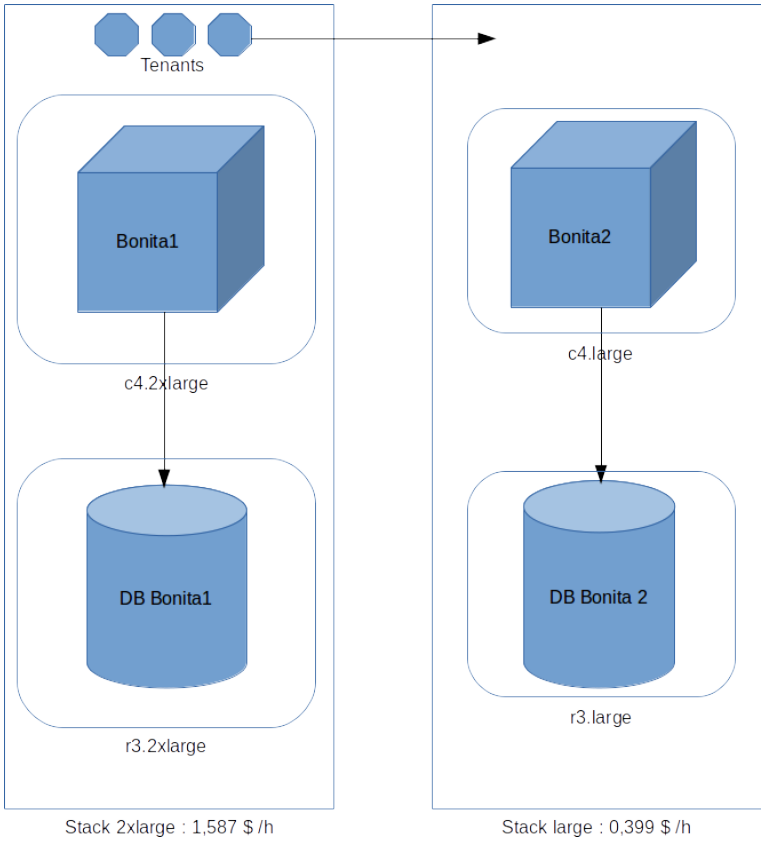

FIGURE 4.8 – Illustration de migrations de tenants d'un système onéreux à un système bon marché

bureau. Comme nous le verrons dans le chapitre 5, une solution simple -et souvent utilisée en production- est l'utilisation d'une ressource capable d'héberger les besoins à tout moment des clients. Toutefois, cette solution reste coûteuse. Comme nous avons pu voir dans la problématique, une solution possible pour pallier ce problème peut être de « déplacer »les installations et données des clients (ou tenants) d'une ressource onéreuse à une ressource moins chère (ou l'inverse dans le cas où les besoins d'un client excéderaient les capacités de sa ressource actuelle).

Nous souhaitons montrer que les migrations ne sont ni des tâches instantanées, ni sans effet sur le comportement des logiciels hébergés. Deux approches existent à ce sujet : effectuer une migration à froid - c'est à dire arrêter le fonctionnement des logiciels concernés avant de copier les données, ou effectuer une migration à chaud (*live migration*) - c'est à dire copier les données durant la période de fonctionnement du logiciel. Notre but étant de minimiser le temps d'indisponibilité des clients, nous considérons dans ce qui suit les migrations à chaud. En effet, durant la période de coupure du système des migrations à froid, les clients n'ont aucune possibilité d'accéder au système hébergé sur la ressource. Ceci se manifeste par des indisponibilités du service pour l'ensemble des tenants s'y trouvant. Toutefois, les migrations à chaud ne sont pas sans effets -même si ceux-ci sont moindres- et il est important d'avoir des moyens d'effectuer ces opérations tout en gardant une cohérence au niveau des traitements et des données persistées du système.

Nous souhaitons mesurer l'impact des migrations à chaud sur la qualité de service du client. Il est assez rare d'avoir des modules permettant de réaliser ce type d'opérations dans les logiciels et plus particulièrement les moteurs BPM. Nous proposons les points suivants :

— une méthode de migrations à chaud peu invasive pour applications Web multi-tenant d'architecture shared-schema (schéma partagé entre les tenants du système) basée sur SQL/MED [23] [100] et l'utilisation de tables externes - décrite dans la première sous-section ;
— une approche générique d'évaluation des migrations à chaud. Son but est de mesurer l'impact des migrations à chaud en termes d'interruption de service et de performances (temps d'exécution) pour les applications Web multi-tenant décrite dans la sous-section suivante ;
— une framework implémentant notre approche ;
— une mesure des impacts de notre méthode de migrations à chaud sur le moteur BPM Bonita 7.4.3 dans sa version Performance.

4.3.1 Besoins en termes d'estimation des effets des migrations de moteurs BPM multi-tenants

Initialement, le terme « migration à chaud »est utilisé dans le cadre de « déplacements »de machines virtuelles à l'intérieur de centre de données d'une ressource matérielle à une autre, sans arrêt des machines virtuelles. L'avantage de ce type d'approche est évident : il rend possible la libération de ressources physiques qui peuvent être utilisées à d'autres fins, ou simplement éteintes. Avec la réduction de la consommation engendrée, cette technique est particulièrement utile dans le contexte dans le cadre de l'informatique verte [101]. La migration de machine virtuelle implique de copier l'état de la mémoire de la machine virtuelle d'origine vers la nouvelle. Nous ne considérons pas ici la migration du stockage correspondant aux machines virtuelles. Les méthodes traditionnelles de migration à chaud font intervenir trois phases [33] :

— une phase de « push »où la machine virtuelle d'origine continue son exécution, pendant que les pages mémoire sont copiées à travers le réseau vers la machine virtuelle de destination. La mémoire modifiée durant ce laps de temps sur la machine virtuelle d'origine sera copiée durant une des phases suivantes ;
— une phase d'arrêt et de copie (*stop-and-copy*) où la machine virtuelle d'origine est arrêtée, et la machine virtuelle de destination est lancée, et le reste des données est copiée ;
— une phase de « pull »où les données restantes sont copiées depuis la machine d'origine vers la machine de destination.

Les méthodes de migrations de machines virtuelles font habituellement intervenir une ou plusieurs de ces phases. L'approche que nous proposons est constituée exclusivement d'une étape « stop-and-copy ». Celle-ci a pour avantage la simplicité et la minimisation de la durée de migration. Toutefois, elle provoque une durée d'indisponibilité plus élevée.

Il existe de nombreux travaux pour les bases de données multi-tenant [10], [102], [45], toutefois ceux-ci nécessitent une adaptation de la base de données et sont intrusives à ce niveau. Comme le précise [102], des opérations simples de *stop-and-copy* tels que l'utilisation d'outils de sauvegarde et récupération comme mysqldump sont envisageables, toutefois ceux-ci agissent sur un schéma entier et ne considèrent pas l'existence de tenants. A notre connaissance, il n'existe aucune technique permettant d'effectuer des migrations de tenants d'une ressource d'origine à une ressource de destination sans coupure du service. Ceci est valable également pour les tenants de bases de données en tant que service. Peu d'approches s'appuient sur le concept de tables étrangères via les Foreign Data Wrapper. On peut citer [103] ou [104] proposant un dispositif et une méthode de base distribuée s'appuyant sur les tables étrangères. Zheng et al. [56] proposent une

23. Management of External Data

approche basée sur un proxy de source de données pour l'application d'architecture multi-tenant sur une application mono-tenant. Même si ces travaux ne ciblent pas les systèmes multi-tenants, ils montrent l'utilisabilité du concept, celui-ci étant également utilisé en production par le site de commerce Alibaba [104].

Nous souhaitons appliquer ce concept à des systèmes multi-tenants. Ceci signifie migrer les données d'un tenant tout en ayant le moins d'impacts possibles sur les performances et la cohérence des données de ce tenant, mais aussi sur celles des autres tenants hébergés sur les systèmes concernés : le système d'origine (là où le tenant que nous souhaitons déplacer est hébergé) et le système destination (là où nous souhaitons déplacer ce tenant).

Pour rappel, plusieurs architectures de systèmes multi-tenant existent, plus précisément *shared table*, *shared process*, et *shared machine*. L'approche que nous présentons ici vise les BPMS multi-tenants avec architecture shared-table. Nous la présentons dans la sous-section suivante.

4.3.2 Une approche de base distribuée et de migrations de tenant pour applications shared table multi-tenant

Nous pouvons décrire une migration par les étapes suivantes, qui doivent être réalisées dans l'ordre indiqué :

1. créer les nouvelles ressources nécessaires au bon fonctionnement (étape nécessaire dans le cloud avec des ressources à la demande)
2. déployer les logiciels nécessaires sur les nouvelles ressources
3. copier les données des utilisateurs depuis les anciennes ressources vers les nouvelles ressources
4. effectuer les opérations nécessaires pour que les requêtes des clients parviennent aux nouvelles ressources en lieu et place des anciennes
5. libérer les anciennes ressources (étape nécessaire dans le cloud avec des ressources à la demande)

L'étape 1 consiste à créer les ressources IaaS nécessaires (ressources de calcul/machines virtuelles, espaces de stockage, interfaces réseau, initialisation du système d'exploitation, démarrage des machines virtuelles concernées) destinées à héberger le tenant migré. Ces opérations peuvent être relativement longues [24]. Dans le cadre d'installation on-premises, ou de configurations statiques cette étape est à ignorer. Ceci est également le cas si l'installation « destination »existe déjà.

L'étape 2 consiste à déployer les logiciels, bibliothèques et réglages statiques et dynamiques [25] nécessaires au bon fonctionnement du système. Ceci peut passer par le déploiement de paquetages, de serveurs d'applications, ainsi que par le paramétrage des liens entre les différentes instances tel que l'adresse de la machine contenant la base de données servant pour la persistance du système. Des outils comme Ansible ou Docker peuvent aider à effectuer ces opérations.

L'étape 3 consiste à copier les données nécessaires au bon fonctionnement du tenant de l'installation d'origine à l'installation destination. Ceci peut se faire de plusieurs manières différentes (comme par le biais de l'API du système), toutefois nous nous concentrons ici sur l'utilisation de scripts SQL. En effet, cette solution permet de récupérer l'état exact dans lequel se trouve le système. Comme nous le verrons dans la description, ces scripts doivent être lancés sans que des opérations soient lancées en parallèle dans le système.

24. Nous avons pu constater des temps d'initialisation pouvant aller jusqu'à une dizaine de minutes autant chez AWS que chez Azure

25. Au titre des ressources physiques utilisées dans le cloud

L'étape 4 consiste à rediriger les appels du client vers la nouvelle installation. L'ensemble des requêtes HTTP précédemment dirigées vers l'installation d'origine doivent parvenir sur la nouvelle installation du tenant. Ceci peut être réalisé par le biais de changement de nommage DNS, ou d'une reconfiguration de l'éventuel équilibreur de charge recevant les requêtes.

L'étape 5 consiste à supprimer les ressources IaaS n'étant plus nécessaires (ressources de calcul/machines virtuelles, espaces de stockage, interfaces réseau, etc.). Ceci n'est envisageable que dans le cas où l'installation n'héberge plus de tenants.

Nous définissons les besoins suivants :

— *parcimonie des données* : seules les données nécessaires et suffisantes au bon fonctionnement doivent être copiées. Ceci est important dans le but de minimiser le temps de migration ;
— *fiabilité* : les données et traitements du tenant doivent rester intègres à chaque moment de la migration ;
— *transparence* : les données doivent être copiées avec un minimum d'effets sur le tenant concerné, et les tenants « colocalisés » ;
— *non intrusion* : le code source du BPMS ne doit pas être modifié, au-delà de modifications de la structure de la base de données et de correction de bugs éventuels. Ceci permet d'éviter d'avoir à recompiler le programme et rend plus simple l'utilisation.

Afin de répondre au besoin de parcimonie des données, nous avons décidé de découper les tables du logiciel concerné en plusieurs catégories, correspondant à l'utilité de leur déplacement :

— tables contenant des données actives ;
— tables contenant des données d'archive ;
— tables contenant des données structurelles.

Nous prenons le parti que seules les données actives sont à déplacer. Celles-ci correspondent aux tables dont les valeurs peuvent être encore modifiées, et qui sont liées aux tenants (par exemple : définitions de processus, instances de processus en cours de traitement, tâches en cours de traitement, évènements en attente, etc.). Les données d'archive correspondent aux traces d'exécution n'étant plus modifiées. Celles-ci concernent les actions réalisées précédemment, mais ne sont accessibles qu'en lecture seule. La taille de ce type de tables peut augmenter progressivement au fur et à mesure du temps. Nous avons considéré que le déplacement de ces données n'est pas nécessaire dans notre cas, et peut être hébergé sur une ressource différente. Le dernier type de table concerne des données structurelles globales à l'installation, n'étant pas forcément liées aux tenants. On peut citer comme exemple la table contenant la liste des tenants, ou également d'autres telles que l'éventuelle table contenant les séquences. Les bonnes pratiques recommandant de séparer les données d'archive des données actives, nous avons cherché un moyen de stocker ces dernières sur le serveur de bases de données du BPMS, tout en stockant les données d'archive et structurelles sur un serveur de bases de données différent, dans notre cas commun à l'ensemble des installations considérées dans le système.

Afin de répondre aux besoins que nous avons énoncé, nous avons proposé une architecture basée sur le concept de *Foreign Data Wrappers*, initialement proposé dans le standard ISO/IEC 9075-9 :2008[19] (également nommé SQL/MED [26]) révisé en 2016 par le standard ISO/IEC 9075-9 :2016. Le principe est de proposer des extensions au langage SQL dans le but de pouvoir accéder à des données externes, par le biais de wrappers et de liens de données. Ce standard a été implémenté par plusieurs fournisseurs de bases de données tels que Postgresql [27], MySQL [28],

26. SQL/Management of External Data
27. `https://wiki.postgresql.org/wiki/Foreign_data_wrappers`
28. `https://dev.mysql.com/doc/refman/5.6/en/federated-create-server.html`

Microsoft SQL Server, IBM DB2, ainsi que d'autres fournisseurs de bases de données moins connus. Le principe passe par l'utilisation par le serveur de base de données de composants logiciels servant de connecteurs et capables de représenter sous la forme de tables l'élément extérieur concerné. Si nous prenons l'exemple de Postgresql, de nombreux connecteurs existent, comme pour la connexion à un autre serveur Postgresql, mais également à d'autres fournisseurs SQL, NoSQL ou même à un système de fichiers ou à des Web Services. Le résultat de cette liaison se manifeste sous la forme de tables externes, se composant de la même manière qu'une table locale [29].

L'avantage de cette technique est qu'elle permet de répondre aisément à notre besoin de non intrusion dans le code source du BPMS, les tables externes se comportant comme des tables conventionnelles. Nous présentons un résumé de l'architecture choisie dans la figure 4.9.

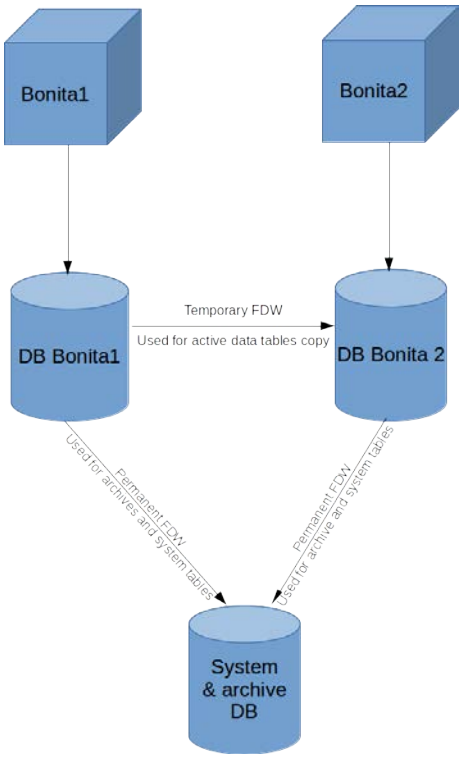

FIGURE 4.9 – Illustration de l'architecture proposée

Dans cette figure, on peut retrouver deux installations du moteur BPM Bonita, une installation d'origine (*Bonita1 / DB Bonita 1*), et une installation de destination (*Bonita2 / DB Bonita 2*). Une instance de base de données référence, destinée à héberger les données de persistance système et d'archive, est ajoutée au système. Des Foreign Data Wrapper permanents permettent de représenter les tables de cette dernière sous forme de tables externes dans les instances DB Bonita1 et DB Bonita 2. Pour des raisons de simplification, nous avons également utilisé un Foreign Data Wrapper -temporaire pour celui-ci- pour la copie de l'ensemble des données entre les deux installations.

La copie des données sans contrôle au niveau des installations peut avoir des effets négatifs sur l'intégrité des données du tenant copié. Il faut parvenir à empêcher l'accès aux tables durant la copie des données, que ce soit en lecture ou en écriture, ceci sans impact sur le code source

29. Certaines limitations existent selon les connecteurs

des BPMS. Dans notre cas, afin de répondre à ce besoin de consistance et de fiabilité, nous nous sommes appuyés sur des fonctions internes des outils concernés permettant de mettre en attente les fonctions du tenant ou de ses composants. Cette étape est importante pour le bon fonctionnement du système, et à régler avant de pouvoir utiliser cette méthode convenablement.

Soit l'installation d'origine A et l'installation de destination B. Les étapes que nous proposons pour la copie sont les suivantes :

— arrêt du tenant ou de ses composantes sur l'installation A ;
— création d'un Foreign Data Wrapper temporaire sur l'installation A pointant vers le schéma complet de l'installation B ;
— copie des données du tenant concerné de l'installation A vers l'installation B ;
— destruction des données du tenant sur l'installation A ;
— destruction du Foreign Data Wrapper ;
— démarrage du tenant ou de ses composantes sur l'installation B.

L'étape de destruction des données du tenant migré sur l'installation d'origine avant le démarrage de celui-ci sur l'installation de destination est importante. Même s'il serait envisageable de déclencher les destructions de manière différée (dans le but d'accélérer la migration), ceci n'est pas possible : les données ayant trait aux tenants faisant partie des données structurelles stockées sur le serveur de référence, l'activation d'un tenant est valable pour l'ensemble des installations. Or, dans une instance BPM, il est possible d'avoir des traitements temporisés se déclenchant de manière différée ou périodique. La présence de ces données sur plusieurs instances risquerait de provoquer une exécution sur plusieurs instances de la même tâche. Imaginons un tenant contenant des instances actives d'un processus métier composé d'une tâche avec un évènement exécuté toutes les 10 secondes. Une fois le tenant déplacé sur l'installation B, les interactions ne se font plus au niveau de l'installation A. Toutefois, l'activation du tenant serait lancée sur les deux installations, l'évènement temporisé serait alors déclenché à nouveau, pouvant engendrer l'exécution de fragments de processus à plusieurs reprises.

Limitations et conclusion

Nous avons présenté dans cette sous-section notre proposition d'architecture et notre approche de migration à chaud pour applications Web multi-tenant d'architecture shared-schema.

Dans l'état actuel de notre approche, quelques limitations sont à considérer, telle que la latence correspondant aux communications entre la base de données de référence et celle de chaque installation. Dans l'état actuel, il est préférable que la base de données de référence soit localisée dans le même centre de données que les installations de moteurs BPM. Dans l'état actuel, nous n'avons pas mesuré cette latence qui sera étudiée dans de futurs travaux. De plus, pour pouvoir faire fonctionner cette technique, nous avons dû retirer les contraintes d'intégrité correspondant aux liens entre les tables de références et les tables de données actives. Ceci n'a toutefois pas posé de problèmes au niveau de l'exécution des processus. Le dernier point que nous avons constaté est une perte de connexion pouvant survenir en cas d'inactivité du système durant plus de 10 minutes. Celle-ci n'a pas eu d'effet sur nos tests, mais est à considérer lors de l'adaptation.

En dehors de ces points qu'il est possible d'adresser, notre approche est pleinement fonctionnelle et est utilisée dans les expérimentations de la section 4.3.5. Elle est de faible empreinte pour les applications, nécessitant seulement la modification du schéma. Moyennant quelques adaptations elle peut être également applicable à n'importe quelle application multi-tenants d'architecture shared-process dont les données actives et d'archive sont pleinement séparées. Nous avons utilisé notre approche pour estimer les effets des migrations à chaud, et décrivons notre technique

d'estimation dans la sous-section suivante.

4.3.3 Résumé de notre approche d'évaluation de l'impact des migrations

Dans cette section, nous présentons notre approche d'évaluation des impacts des migrations à chaud sur la durée d'interruption de service, et des performances des tenants migrés et hébergés sur les ressources concernées. Nous décrivons en premier les hypothèses que nous avons employées pour définir cette approche. Enfin, nous décrivons l'approche que nous proposons pour répondre à cette problématique, de manière répétable pour les applications Web incluant serveurs d'applications et bases de données.

Mesure de l'impact des migrations

Nous partons du principe que durant une migration à chaud, les clients des services hébergés sur les ressources d'origine et de destination verront une dégradation de leurs performances. Afin de mesurer ces impacts, nous souhaitons utiliser les trois métriques suivantes.

— La durée de migration (ou durée d'interruption de service) est la durée pouvant être mesurée entre le début de la migration sur la ressource d'origine, soit le moment où le tenant cesse de recevoir des requêtes, et la fin de migration soit le moment où elle peut commencer à recevoir des requêtes sur la ressource de destination. Nous sommes intéressés par la caractérisation des facteurs influençant cette durée, comme par exemple la taille des données à migrer. Cette quantité de données est proportionnelle au nombre d'éléments présents dans la base de données de l'application concernée. Nous nommons cette première métrique : « ***durée de migration*** ».

— La seconde métrique mesure les impacts des migrations sur les performances du tenant migré. Nous proposons de comparer le temps de traitement et de réponse du tenant migré avant et après migration. Nous nommons cette métrique « ***effets des migrations sur le tenant migré*** ».

— La troisième métrique considérée concerne les performances des opérations des tenants hébergés sur les ressources d'origine et de destination durant la migration. Les ressources de calcul, mémoire et réseau nécessitées par les migrations peuvent avoir des effets sur les autres tenants. Nous nommons cette métrique « ***effets des migrations sur les tenants colocalisés*** »

Nous souhaitons mesurer le temps de calcul des opérations du système selon deux différentes perspectives : celle du client, et celle du fournisseur de services. En effet, selon les systèmes, la réponse d'une requête côté client n'implique pas la fin des opérations associées dans l'application concernée. Par exemple, pour un BPMS, le moteur peut se retrouver à prendre un temps supplémentaire pour terminer l'opération après la satisfaction de la requête du client. En effet, il peut y avoir des tâches supplémentaires dans le cadre d'évènements temporisés comme nous avons précisé plus tôt. L'ensemble de ces mesures doit être stockée sous une forme normalisée, afin de pouvoir comparer facilement les performances entre différents scénarios de tests.

Hypothèses sur l'impact des migrations de tenants

Nous énonçons ici plusieurs hypothèses sur les effets négatifs des migrations sur la durée d'interruption de services et les effets sur les performances. Les hypothèses que nous souhaitons vérifier sont les suivantes :

— Les *durées de migrations* sont prédictibles et augmentent avec la taille des données à migrer. Les migrations consistent à copier les données de l'application Web depuis la

ressource d'origine vers la ressource de destination. Plus de données signifie plus de temps pour la copie, vu que celles-ci doivent être lues depuis la ressource d'origine, copiées à travers le réseau, et écrites sur la ressource de destination.

— Les *effets des migrations sur le tenant migré* existent et sont provoquées par la création du nouveau tenant sur la ressource destination. Ceci pourrait avoir plusieurs causes comme par exemple les défauts de cache ou l'initialisation asynchrone de certaines librairies.

— Les *effets des migrations sur les tenants colocalisés* existent et doivent être pris en compte. Les différentes étapes des migrations auront des effets sur les performances des ressources d'origine et de destination, et sur l'ensemble des tenants hébergés sur celles-ci. En effet, l'activation et la désactivation de tenants, ainsi que la copie des données à travers le réseau nécessitent de la puissance de calcul, et de la bande passante de stockage et réseau qui vont provoquer des temps de calcul plus élevés pour les requêtes réalisées sur les tenants concernés.

Processus de mesure

Afin de pouvoir mesurer ces différentes métriques, nous avons simulé les utilisateurs interagissant avec les applications Web testées. Ceci passe par l'émulation des différentes requêtes HTTP que les utilisateurs devraient exécuter pour récupérer les tâches, se les assigner, et les traiter. Pour chaque mesure, nous voulons observer des variations, telles que la nature de la charge ou la quantité d'opérations initialisée dans le système. Nous souhaitons archiver la durée de chaque opération exécutée sur l'application Web, ainsi que celle de chaque migration effectuée. Il est alors envisageable de comparer les résultats statistiquement.

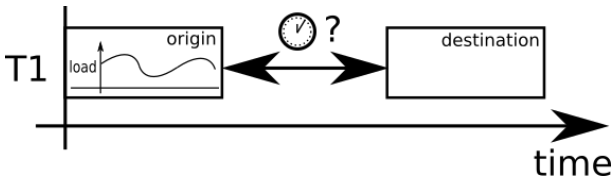

FIGURE 4.10 – Illustration de l'estimation de la durée de migration

Durée de migration Nous proposons une première expérimentation où nous exécutons une charge de travail définie pour un tenant, hébergé seul sur la ressource d'origine. Nous migrons alors ce tenant depuis la ressource d'origine vers la ressource de destination, et observons la durée de migration correspondante, ainsi que ses composantes internes. La figure 4.10 illustre cette expérimentation.

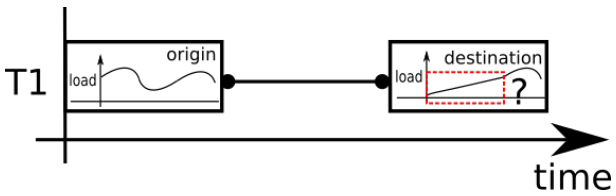

FIGURE 4.11 – Illustration de l'estimation des effets sur le tenant migré

Effet des migrations sur le tenant migré Dans cette expérimentation, nous proposons d'effectuer les différentes tâches réalisées dans l'expérimentation « *Durée de migration* », à laquelle nous ajoutons l'exécution d'une charge de travail sur la ressource de destination, suite à la migration. Afin de mesurer ces effets, nous mesurons les performances du tenant sur la ressource d'origine et sur la ressource de destination, et nous les comparons. Nous pouvons récupérer les temps de traitement au niveau de la base de données de l'application Web, ainsi que les temps de réponse au niveau du simulateur de client. Chaque horodatage doit être stocké, pour chaque étape, ainsi que pleinement identifiée à l'aide d'un identifiant de test afin de pouvoir comparer les différents points de vue. La figure 4.11 illustre cette expérimentation.

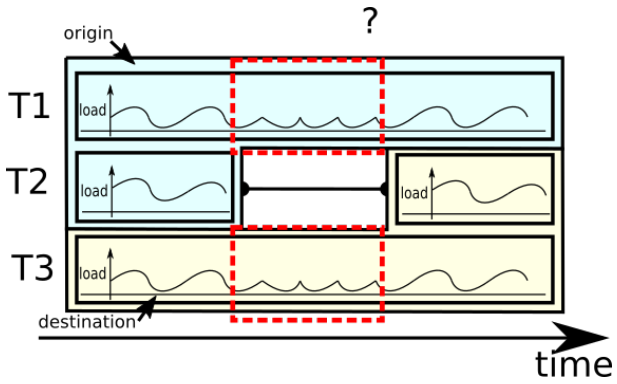

FIGURE 4.12 – Illustration de l'estimation des effets des migrations sur les tenants colocalisés

Effets des migrations sur les tenants colocalisés Nous souhaitons comparer les effets d'une migration sur les tenants hébergés sur la ressource d'origine et de destination. Dans ce cas nous proposons de déclencher des charges de travail de longue durée sur les tenants des ressources d'origine et de destination. Nous exécutons aussi une charge de travail courte sur le tenant destiné à être migré, de manière similaire aux expérimentations précédentes. Après sa migration, une charge de travail courte est alors exécutée sur le tenant migré sur la ressource de destination.

La figure 4.12 présente un exemple de scénario, avec trois tenants : $T1$ est hébergé sur la ressource d'origine, $T3$ sur la ressource de destination. $T2$ est tout d'abord hébergé sur la ressource d'origine, puis migré sur la ressource de destination. Nous voulons savoir comment les performances des requêtes réalisées par les clients des tenants $T1$ et $T3$ sont affectées par la migration, comparée à un moment similaire sans migration.

Nous considérons comme "requête concernée par une migration" chaque requête en cours d'exécution pour entre le début de la migration, et la fin de la migration, soit où l'horodatage du début de requête est inférieur à celui de la fin de la migration, ou l'horodatage de la réponse à la requête est supérieur à celui du début de la migration. Nous comparons alors les durées de calcul et les durées de requêtes pour des moments où la charge de travail totale est similaire. La différence entre les durées de calcul nous montrera les effets de la migration.

4.3.4 Framework d'évaluation de migrations pour BPMS multi-tenant

Afin de tester notre approche, nous avons développé un framework de test permettant d'évaluer l'impact des migrations pour les BPMS multi-tenants. Celui-ci prend en entrée une série de paramètres (décrite plus bas dans le tableau 4.4) et fournit en sortie une série de mesures

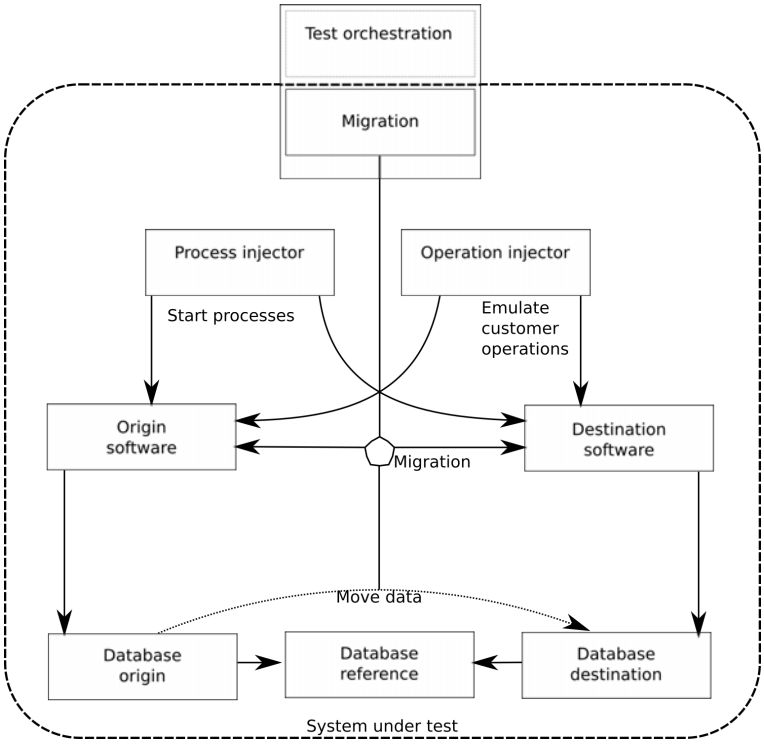

FIGURE 4.13 – Architecture du framework d'évaluation de migrations pour BPMS multi-tenant

concernant les horodatages des migrations et des exécutions de tâches, et de processus BPM, du point de vue du client et du moteur BPM. La figure 4.13 illustre l'architecture du framework proposée. On peut y distinguer les éléments suivants :

— le composant de migration utilisé pour déplacer les tenants d'une installation à l'autre (tel que décrit dans la section 4.3.3),
— les composants de simulation de clients utilisés pour générer les charges de travail du système testé,
— le système testé représente le BPMS étudié.

Composants

Le but premier des composants nommés « load testing »dans la figure 4.13 est d'émuler le comportement d'utilisateurs de tenants interagissant avec des BPMS. Nous avons utilisé deux catégories de composant pour cette émulation :

— un testeur de charge dont le but est d'initialiser les tenants, les schémas BPM, les utilisateurs et leurs dépendances, puis de déclencher de nouveaux processus,
— un module "agent BPM" dont le but est d'émuler les utilisateurs en charge des tâches des processus, basé sur les systèmes multi-agents (MAS) [105].

Injecteur de processus Nous avons utilisé une version spécifique du testeur de charge Faban [30] préparée dans le framework Benchflow[97]. Cette version a été modifiée afin de pouvoir être exécutée dans des conteneurs Docker, et fournit également des API supplémentaires permet-

30. `http://faban.org/about.html`

tant d'automatiser les interactions. Celles-ci se présent sous la forme d'images Docker [31] et d'un client HTTP Java permettant le déploiement de scénarios et l'exécution de charges de travail par le biais d'une invite de commande. Deux images Docker ont été utilisées dans notre cas : l'image Faban Harness qui correspond à un contrôleur, et l'image Faban agent correspondant à un agent d'exécution. Il est possible d'utiliser plusieurs agents Faban pour un contrôleur, afin d'assurer le passage à l'échelle du testeur de charge.

Les différentes étapes déclenchées par les scénarios doivent être les suivantes :

— Création des tenants concernés et des utilisateurs, puis déploiement du schéma BPM destiné à être testé, et ajout des autorisations et rôles nécessaires pour le bon fonctionnement.

— Démarrage des instances de processus selon le scénario déterminé, par appels à l'API HTTP du BPMS.

BPM agents Un processus métier implique l'exécution d'une séquence de tâches reliées qui sont exécutées dans un ordre donné afin d'accomplir un but. Certaines de ces tâches sont automatisées, d'autres sont préparées pour être exécutées par des ressources humaines, habituellement à travers un formulaire. Une ressource humaine est un utilisateur ayant différentes capacités et compétences qui déterminent son rôle dans l'exécution du processus. Un tenant de BPMS a une structure organisationnelle qui consiste en différentes groupes d'utilisateurs. Selon leurs profils et rôles, les utilisateurs peuvent traiter différentes activités à l'intérieur de leur organisation.

Nous avons utilisé un système multi-agents (MAS) pour modéliser et simuler les utilisateurs impliqués dans l'exécution des activités business de chaque tenant. Un MAS est constitué d'un certain nombre d'agents dans un environnement commun (réel ou virtuel) où ils peuvent agir et coopérer afin d'arriver à un objectif. Un certain nombre de travaux de recherche ont utilisé les MAS pour modéliser et simuler des processus métiers en proposant une correspondance entre des éléments BPMN et MAS [106], [107], [108]. Avec leurs capacités en autonomie, flexibilité et adaptabilité, nous partons du principe que les agents représentent des solutions efficaces pour modéliser et simuler le comportement des ressources humaines. Dans ce module, nous modélisons les utilisateurs des tenants en tant qu'agents. Leur but est de réaliser les tâches des processus métiers afin de réaliser l'exécution des processus contenant des tâches humaines et de garantir un niveau stable de tâches actives durant la migration des tenants, afin d'étudier leur comportement.

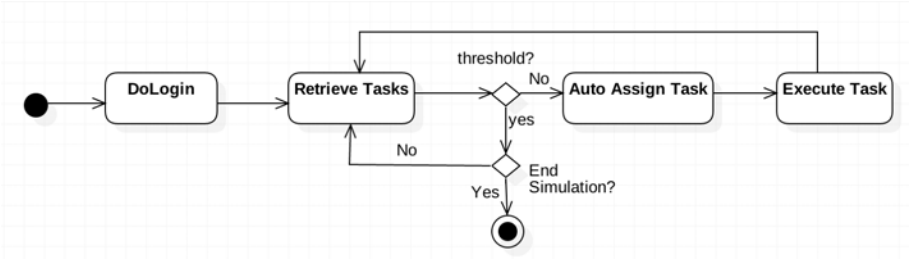

FIGURE 4.14 – Diagramme d'état décrivant le comportement des agents

Chaque agent représente une entité active réalisant des tâches pour l'utilisateur d'un tenant. Comme le montre la figure 4.14, décrivant le diagramme d'état d'un agent, ce dernier commence par se connecter à la plateforme BPMS. Ensuite, il récupère les tâches disponibles (non attribuées). Si le nombre de tâches disponibles est supérieur à un seuil donné, l'agent s'assigne la tâche, puis l'exécute. Dans le cas contraire, l'agent ne fait rien jusqu'à ce que des instances de

31. https ://github.com/benchflow/docker-images/tree/dev/faban

processus soient actives. Le but du comportement basé sur les seuils de quantité de tâches est de permettre d'étudier le comportement des migrations selon le nombre de tâches humaines actives. Nous donnons plus de détails sur cet aspect dans la section suivante.

Afin de pouvoir intégrer facilement ce module à notre framework, nous avons préparé une image Docker [32]. Celle-ci supporte les paramètres nécessaires pour le bon fonctionnement comme par exemple l'adresse du BPMS, les identifiants de l'utilisateur concerné, le seuil en nombre de tâches, etc.

Migrations Afin de déclencher les migrations à chaud, nous avions besoin d'implémenter notre méthode, décrite dans la section 4.3.2. Nous l'avons fait à l'aide de scripts Python et de requêtes SQL. La fonction des scripts Python est d'orchestrer la migration (arrêt du tenant sur la ressource d'origine, copie des données depuis la ressource d'origine vers la ressource de destination et activation du tenant sur la ressource de destination). La copie des données se fait au travers de scripts SQL exécutés par le script Python, à l'aide de la bibliothèque Psycopg [33]. Comme pour les deux modules précédemment décrits, nous avons préparé une image Docker avec comme paramètres les besoins de ces scripts, plus précisément :

— l'adresse et les identifiants des moteurs BPM d'origine et de destination,
— l'adresse et les identifiants de leur base de données respectives,
— l'identifiant du tenant que l'on souhaite migrer,
— le type de BPMS concerné.

Dans l'état actuel, deux BPMS sont disponibles dans cette image Docker : BonitaBPM 7.4.3 (nécessite l'utilisation de la version commerciale Performance de Bonita) et Camunda 9.6.0.

Automatisation des tests

Nous utilisons Docker pour le déploiement et le lancement du système étudié, pour l'exécution des scripts d'injection de processus et de réalisation de tâches, ainsi que pour l'exécution de la migration. Le test de solutions logicielles distribuées sur différentes ressources peut être facilité par l'utilisation d'orchestrateurs. Pour cela, nous proposons l'utilisation de Docker Swarm [34] pour la gestion du cluster, et Docker Compose pour la description des systèmes distribués. Docker Swarm est une fonctionnalité intégrée de Docker dont le but est de gérer des conteneurs sur un système distribué. Une telle fonctionnalité nous est clairement utile vu que nous souhaitons distribuer les conteneurs sur différentes machines, en l'occurrence une ressource matérielle pour chaque brique logicielle. Un réseau virtuel de type *overlay network* est créé pour faciliter les communications entre conteneurs. Celui-ci consiste en une couche supplémentaire. Chaque pile logicielle déployée est nommée à l'aide d'un identifiant, de manière isolée. Docker Compose permet de décrire une suite logicielle composée de plusieurs conteneurs ayant des dépendances, comme par exemple l'utilisation d'une base de données par le moteur BPM. Le BPMS concerné doit donc avoir un fichier Docker Compose permettant de le déployer sur l'infrastructure. Il est ensuite nécessaire de spécifier certaines instruction permettant l'affinité des conteneurs avec les différentes ressource systèmes employées. Dans ce but, chaque nœud est préalablement étiqueté en fonction de sa future fonction (agent BPM, BPMS, base de données, conteneurs Faban), et de son type pour le BPMS testé (origine ou destination). Une fois le déploiement déclenché, les conteneurs sont déployés sur les ressources correspondantes.

32. `https://github.com/Chahrazed-l/AgentBPMS`
33. `http://initd.org/psycopg/`
34. `https://docs.docker.com/engine/swarm/`

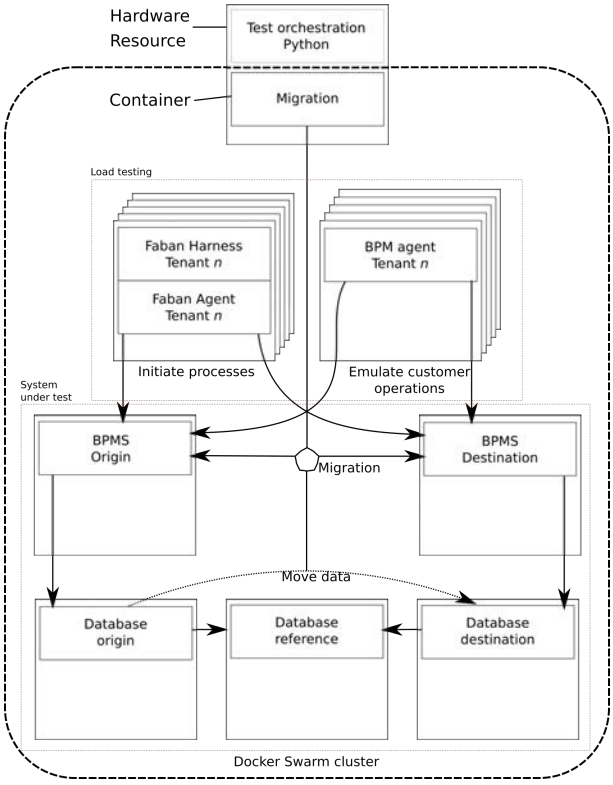

FIGURE 4.15 – Exemple d'architecture du framework de tests de migrations

Nous avons également préparé un fichier Docker Compose pour les modules Faban et agents BPM, s'appuyant sur leurs images Docker respectives. Par ce biais, il est envisageable de déployer facilement plusieurs exemplaires de ceux-ci sur différentes ressources. Ceci peut être intéressant pour assurer la scalabilité de l'injection de processus et de tâches. On peut par exemple prévoir d'avoir une instance de chaque module, déployée sur une ressource spécifique pour chaque tenant. Ceux-ci doivent être déployés sur la même pile que celle du système testé, dans un second temps.

Le schéma 4.15 présente un exemple d'architecture. On y retrouve les différentes ressources utilisées par chaque module. Dans ce cas précis 5 ressources sont employées pour le module Faban, et 5 ressources sont employées pour le module BPM. Les scripts de contrôle (« orchestrator ») et le conteneur effectuant la migration sont exécutés sur la même instance.

Orchestration

Un utilisateur souhaitant utiliser notre framework doit préparer un descripteur contenant la liste des paramètres de l'expérimentation. Celui-ci est à fournir sous la forme d'un dictionnaire Python avec la liste de paramètres souhaités. Ces derniers permettent plus particulièrement de changer le type de BPMS visé, le schéma employé (qui doit être préalablement fourni), ainsi que d'autres paramètres ayant trait à la durée de l'expérimentation. La liste est décrite dans le tableau 4.4. Le lecteur pourra retrouver quelques détails importants sur les paramètres ci-dessous.

— *target* décrit dans un fichier propriété les adresses des instances de BPMS d'origine et de destination. Les instances de l'infrastructure doivent être nommées en conséquence dans le fichier Docker Compose.

— *no_reset_needs* permet d'éviter de remettre à zéro l'initialisation de Docker Swarm, des

Nom	**Catégorie**	**Fonction**	**Exemple**
experiment_type	Identifiant test	Nom du type d'expérimetation	xp3
iteration	Identifiant test	Numéro expérimentation	1
target	Infrastructure	Paramètres nécessaires pour injection	["config_pre.properties", "config_pre.properties", "config_post.properties"]
tenant_ips	Infrastructure	Adresses des agents Faban	["faban-agent1", "faban-agent2", "faban-agent3"]
bpm_ips	Infrastructure	Adresses des agents BPM	["bpm-agent1", "bpm-agent2", "bpm-agent3"]
no_reset_needs	Infrastructure	Remise à zéro	True
bpm	Paramètres test	Nom du schéma BPM	TestHumanTask-
bpms	Paramètres test	Nom du BPMS	bonita
faban_agents	Paramètres test	Nombre d'agents Faban	1
nb_agents	Paramètres test	Nombre d'agents BPM	100
nb_process	Paramètres test	Nombre de processus actifs	50
duration_tenants_bg	Paramètres test	Durée d'injection pour les tenants non migrés	1200
duration_tenants_migr	Paramètres test	Durée d'injection pour les tenants migrés	120
tenants	Paramètres test	Liste des tenants	["tenant1", "tenant2", "tenant3"]
tenants_migr	Paramètres test	Liste des tenants à migrer	["tenant1"]
back_and_forth	Paramètres test	Nombre de migrations	1
duration_before_migration	Paramètres test	Durée avant migration	300

Table 4.4 – Paramètres principaux du framework

images Docker, et des paramètres Faban et agent BPM sur les noeuds du cluster. Ce paramètre, à initialiser à *faux* au premier lancement, est à placer à *vrai* ensuite, si le test concerne le même nombre d'agents émulant les clients.
— *faban_agents* représente le nombre d'agents Faban utilisés pour chaque instance Faban. Ceux-ci représentent le nombre d'injecteurs devant être utilisés.
— *nb_agents* représente le nombre d'agents utilisés pour chaque instance de l'émulateur de clients.
— *nb_process* représente le nombre de processus gardés en base de données par l'émulateur de clients.
— *back_and_forth* représente le nombre de migrations réalisées entre l'origine et la destination durant l'exécution du test. Ces migrations sont réalisées séquentiellement de l'origine à la destination, puis de la destination à l'origine.
— *duration_before_migration* représente le temps d'attente entre la fin du test effectué sur le BPMS d'origine et le déclenchement de la migration. Fixer une valeur suffisante pour cette durée est important, car le déroulement des tâches des processus prend du temps, et dépasse souvent la durée d'injection des processus.
— les paramètres *tenant_ips* et *bpm_ips* permettent de fixer les instances contenant les agents Faban et les agents BPM.

Quand un test est lancé, les étapes suivantes sont déclenchées dans l'ordre :

1. initialisation du test : création d'un identifiant unique pour le test et le nom du répertoire de résultats correspondant à ce test,
2. déploiement des besoins : remise à zéro et initialisation de Docker Swarm et des fichiers nécessaires (images Docker, fichier paramètre, etc.) sur les ressources,
3. déploiement du BPMS : déploiement à l'aide de Docker de l'ensemble des composants à tester (moteurs BPM et bases de données) sur les ressources,
4. lancement du test : déploiement du testeur de charge Faban et des agents BPM suivie des tests,
5. copie des résultats : lancement de requêtes sur la base de données afin de récupérer les durées de calcul des processus et tâches et récupération des fichiers de résultats des agents BPM pour les durées des requêtes correspondantes. L'ensemble de ces résultats est ensuite stocké sous forme de fichiers CSV dans le répertoire de résultats du test.

Le principe général de l'étape 4, illustrée par la figure 4.16, est de lancer de multiples charges sur un ensemble de tenants pour une durée définie, nommés "background tenants" pendant qu'un tenant est chargé sur la configuration d'origine, migré, puis chargé sur la configuration de destination ("migrated tenant"). Les tests sur les "background tenants" sont réalisés de manière asynchrone. Une fois l'ensemble des actions terminées, les résultats des tests peuvent alors être récupérés depuis la base de données, et les agents BPM.

Dans l'étape 5, les résultats que nous souhaitons récupérer consistent en la durée de la migration, le temps de réponse et la durée de calcul de chaque tâche correspondant aux processus lancés durant ce test, ainsi que de la durée des processus. Pour cela, nous stockons l'ensemble des horodatages correspondant à chaque état de processus et de tâche. Ces résultats sont stockés dans un ensemble de fichiers CSV.

4.3.5 Expérimentation

Nous expliquons dans ce qui suit comment nous avons adapté notre framework pour notre cas d'utilisation.

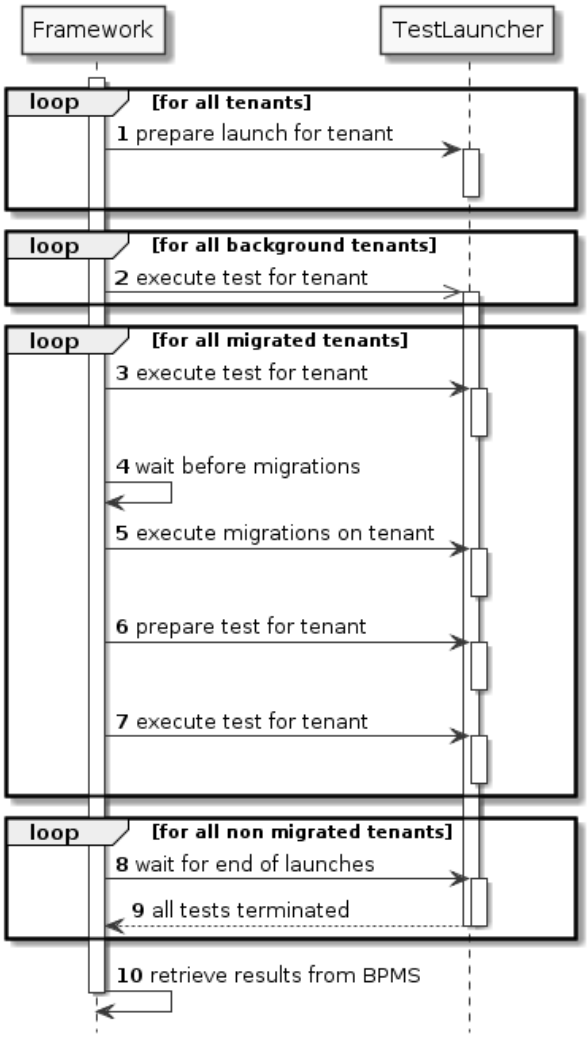

FIGURE 4.16 – Diagramme de séquence décrivant le lancement d'un test

Système testé

Comme nous l'avions montré dans la figure 4.13, le système testé est composé de 5 composants correspondant à l'installation d'origine et de destination, chacune composée d'un BPMS et d'une base de données. A cela s'ajoute la base de référence contenant les données d'archive et de système. Pour nos tests, nous avons utilisé une image basée sur l'image Docker officielle de la solution Bonita 7.4.3 dans sa version commerciale "Performance edition" [35]. Pour la base de données nous avons utilisé l'image officielle de Postgresql [36] dans sa version 10.3.

Détails sur la migration à chaud

Nous avons appliqué notre méthode de migration à chaud décrite dans la section4.3.1 sur le BPMS Bonita. Pour cela, nous avons préparé une image Docker spécifique permettant de supporter la base de données de référence, que nous avons fait hériter à partir de l'image officielle

35. La fonctionnalité multi-tenant n'est disponible que pour cette licence commerciale, non libre.
36. `https://hub.docker.com/_/postgres/`

précédemment indiquée. Pour arriver à préparer cette version modifiée, nous avons réalisé les étapes suivantes :

- classification des tables de la base de données de Bonita en deux catégories : tables de tenant concernant des données actives, et tables concernant des données non actives, ou globales à la plateforme,
- préparation du script de création de la base de référence contenant les tables de données non actives et globales. Celui-ci reprend l'ensemble des requêtes SQL de type "CREATE TABLE" et "ALTER TABLE" du script "createTables.sql" servant à l'initialisation de la base de données du moteur [37],
- modification des scripts de création de base de données de Bonita 7.4.3, avec création d'un Foreign Data Wrapper vers la base de référence, et le remplacement des scripts de création des tables de données non actives et globales par la création de tables externes correspondant aux tables déplacées dans la base de référence. La création du Foreign Data Wrapper nécessite les adresses et identifiants de la base données de référence,
- préparation de l'image Docker étendue, avec comme paramètres supplémentaires l'adresse et les identifiants de la base de données de référence.

Nous avons également préparé une image Docker faisant appel à des scripts SQL de migrations, et des appels à l'API REST de Bonita [38]. Nous précisons ci-dessous l'ensemble des étapes réalisées :

- Préparation des scripts de migration. Le principe de ceux-ci est de créer un Foreign Data Wrapper temporaire sur la base de données du BPMS d'origine pointant sur la base de données du BPMS destination. L'ensemble des tables physiques y est alors déclaré comme tables externes.
- Préparation du script Python d'orchestration. Comme nous l'avons précisé dans la sous-section 4.3.1, celui-ci déclenche une désactivation du tenant sur la configuration d'origine. Les scripts SQL de migration sont ensuite lancés. Puis un appel à l'API REST du moteur BPM de la configuration cible est déclenché pour activer le tenant.

Processus métiers

FIGURE 4.17 – Schéma BPMN TestHumanTask

Dans cette expérimentation, nous avons considéré trois processus métier différents afin d'étudier les effets de la structure du schéma sur la durée de migration. Ces processus ont été modélisés à l'aide du langage BPMN. Nous avons tenu à inclure des tâches humaines dans chacun de ces schémas de processus. En plus de l'intérêt de l'étude de ce type de tâches, avec notre architecture le meilleur moyen de considérer des variations de taille de la base de données du moteur était d'utiliser des tâches humaines. En effet, celles-ci, au contraire des tâches automatiques nécessitent une interaction pour être fermées. L'exécution de tâches automatiques sans traitement est réalisé immédiatement, et lors de la désactivation des tenants l'ensemble des tâches automatiques en

37. La version communautaire du script d'origine est accessible sur ce lien : `https://github.com/bonitasoft/bonita-engine/blob/7.4.3/platform/platform-resources/src/main/resources/sql/postgres/createTables.sql`

38. `https://documentation.bonitasoft.com/bonita/7.4/platform-api`

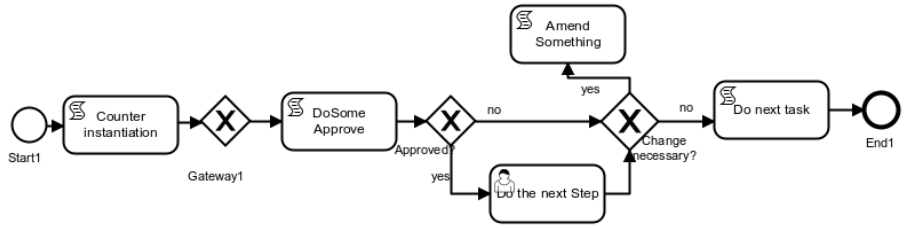

FIGURE 4.18 – Schéma BPMN AdditionalApproval

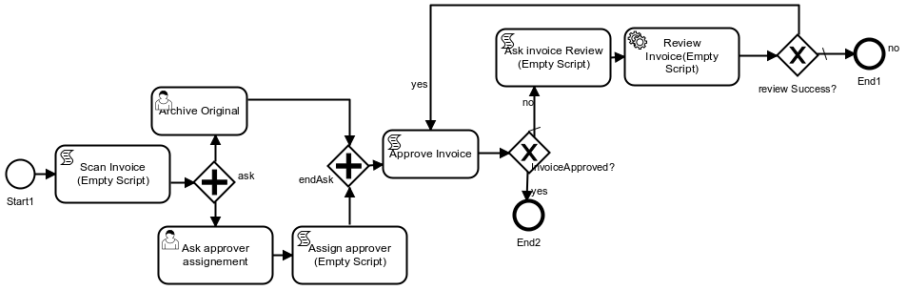

FIGURE 4.19 – Schéma BPMN M3Process

cours est terminée, ce qui ne provoque la migration d'aucune tâche en cours si on ne considère que les tâches automatiques.

Le premier modèle, décrit par la figure 4.17, nommé *TestHumanTask* représente la plus simple des structures. Elle consiste en une seule tâche humaine. Le reste des modèles s'appuie sur un mélange entre tâches humaines et automatiques, branchements exclusifs et inclusifs. Ces derniers ont été initialement définis et utilisés dans [94]. Le deuxième modèle, nommé *AdditionalApproval*, décrit par la figure 4.18 consiste en un ensemble de tâches automatiques, auxquelles on a ajouté une unique tâche humaine. Le processus démarre en initialisant une variable de processus x à 0 (tâche *Counter Instantiation*), qui est incrémentée par la tâche automatique *DoSomeApprove* située après le premier branchement exclusif. Les branchements *Change necessary* dirigent vers la condition *no* si la valeur de la variable x est inférieure à 5, terminant le processus après la tâche automatique *Do next task*. Dans le cas contraire, une tâche humaine est déclenchée, et une boucle ramène au brancement *Gateway1* après l'exécution de la tâche *Amend Something*. Ceci permet d'obtenir un comprotement déterministe. Les tâches automatiques ne modifiant pas la valeur de la variable de processus x n'exécutent aucun traitement. Le troisième et dernier processus testé, *M3Process* a une structure plus complexe. Il consiste en premier lieu à tester le cas de deux tâches humaines exécutées en parallèle. Le processus initialise une variable x à 0. Celle-ci est incrémentée dans la tâche *Approve Invoice*. Le branchement *review Success* effectue une boucle jusqu'à la tâche *Approve Invoice* si la valeur de x est inférieure à 4. Les tâches automatiques ne modifiant pas la valeur de la variable de processus x n'exécutent aucun traitement.

Détails sur l'expérimentation

Nous avons effectué les tests suivants, dans le but de vérifier nos hypothèses de la section 4.3.3.

— *Durée de migration* : étude de la durée des migrations sans interaction durant la migration (chaque processus et tâche est dans un état soit non assigné soit terminé, aucune injec-

tion de nouveaux processus ou fermeture de tâche n'a lieu durant la migration). Nous souhaitons observer la durée en fonction du nombre de processus actifs dans la base de données.

— *Effets des migrations sur le tenant migré* : comparaison de la durée des processus et de tâches du tenant migré avant et après la migration.

— *Effets des migrations sur les tenants colocalisés* : comparaison des durées des processus et tâches pour les tenants non migrés durant une migration, comparé à la même situation quand aucune migration n'a eu lieu.

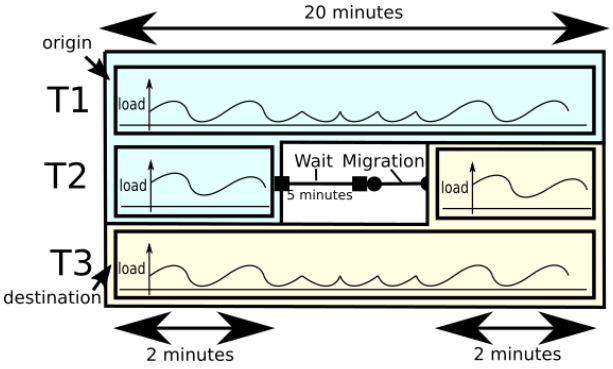

FIGURE 4.20 – Deuxième expérimentation

Afin d'évaluer ces trois tests, nous avons réalisé deux types d'expérimentations :

— Le but de la première expérimentation est d'évaluer la *durée de migration*. Dans ce cas, nous avons lancé le framework pour un seul tenant. Pour chaque processus, nous avons ensuite exécuté une charge pour itérativement 0, 5, 10, 30, 60, 120, 300 et 600 secondes. Après cette charge, nous avons déclenché six migrations à la suite, itérativement du BPMS d'origine à la destination, et de la destination à l'origine. Nous avons alors observé la durée totale de la migration, ainsi que des détails sur ses étapes internes : désactivation du tenant, migration des données, et activation du tenant.

— Le but de la deuxième expérimentation est d'évaluer les *effets des migrations sur le tenant migré* et les *effets des migrations sur les tenants colocalisés*. Celle-ci est illustrée par la figure 4.20. Dans ce cas nous avons considéré trois tenants. Deux tenants sont lancés en arrière-plan, et jouent le rôle de tenants colocalisés : un simulé sur la configuration d'origine (*tenant1*), un simulé sur la configuration de destination (*tenant3*). Chacun de ces tenants a une injection de 20 minutes par le module Faban, pendant que 100 agents BPM pour chaque tenant clôturent leurs tâches. Un troisième tenant (*tenant2*) est d'abord déclenché sur la configuration d'origine durant 2 minutes. Après 5 minutes de temps d'attente, la migration est déclenchée et une charge de 2 minutes est alors effectuée pour celui-ci, sur la configuration de destination.

Infrastructure de tests

Nous avons exécuté nos tests à l'aide du fournisseur IaaS Azure Public Cloud. Nous avons utilisé les types d'instances suivants :

— Bases de données : Standard E2s v3 (2 vcpus, 16 GB memory) - 3 instances

— BPMS : Standard F4s (4 vcpus, 8 GB memory) - 2 instances

— Faban testeur de charge (Harness et Agent) : Standard F1s (1 vcpus, 2 GB memory) - 1 et 3 instances respectivement pour la première et la deuxième expérimentation

— Agents BPM : Standard F2s (2 vcpus, 4 GB memory) - 0 et 3 instances respectivement pour la première et la deuxième expérimentation
— Orchestrateur : Standard B2ms (2 vcpus, 8 GB memory) - 1 instance

La famille d'instance E-series [39] est optimisée pour la mémoire. Nous l'avons utilisée pour les bases de données. La famille d'instance F-series [40] est optimisée pour le calcul. Nous l'avons utilisée pour le moteur BPM, les agents BPM, et le testeur de charge Faban. Le type d'instance que nous avons utilisé pour l'orchestration et la récupération des données est à capacité variable (famille B-series). Nous l'avons choisie pour son faible coût.

4.3.6 Résultats numériques obtenus et discussions

Nous décrivons dans cette sous-section les résultats que nous avons obtenu.

Durée des migrations

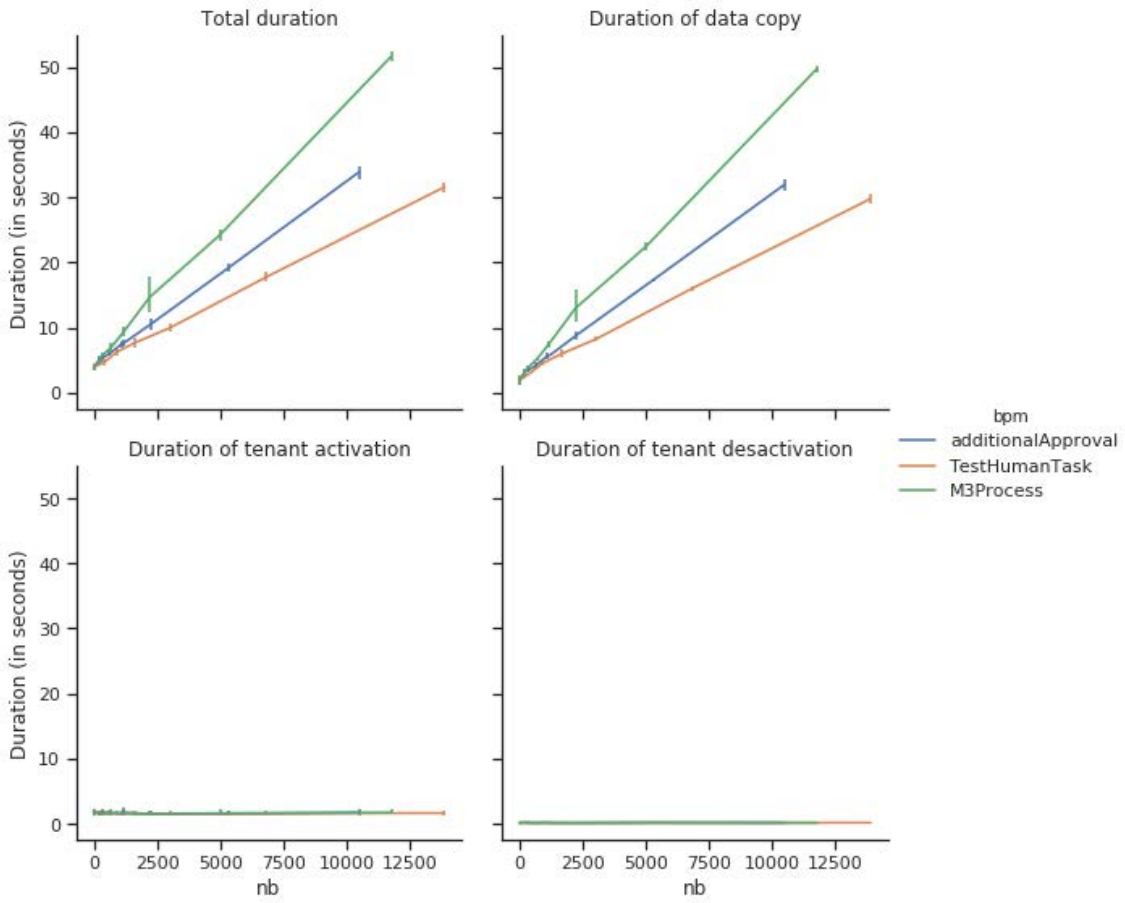

FIGURE 4.21 – Durée des migrations pour chaque schéma BPM en fonction du nombre de processus

39. Standard memory optimized instance :https://docs.microsoft.com/en-us/azure/virtual-machines/linux/sizes-memory
40. Standard computing optimized instance : https://docs.microsoft.com/en-us/azure/virtual-machines/linux/sizes-compute

Dans la figure 4.21, nous avons décrit la durée totale de migration ainsi que ses 3 composantes : dans l'ordre la durée de copie, la durée d'activation, et la durée de désactivation, pour les trois types de processus.

On peut remarquer que la désactivation des tenants est très rapide (quelques dizaines de millisecondes), et stable quel que soit le nombre de processus actifs et le schéma BPM étudié. L'activation des tenants est moins stable et dure en général quelques secondes. C'est la durée la plus significative quand on ne migre aucun processus. La copie des données est très linéaire en fonction du nombre de processus. Elle est plus rapide pour le schéma le plus simple, *TestHumanTask* (environ 20 secondes pour 10000 processus métiers). La copie des données du schéma *AdditionalApproval* prend plus de temps (environ 30 secondes pour 10000 processus), et celle du schéma *M3Process* presque 50 secondes. On peut remarquer un peu de variabilité au niveau du processus de copie.

La variabilité de l'activation comparée à celle de la désactivation nous a amené à étudier les causes de celles-ci. On peut retrouver dans la figure 4.22, qui représente les durées d'activation et de désactivation en fonction de l'ordre de lancement de la migration.

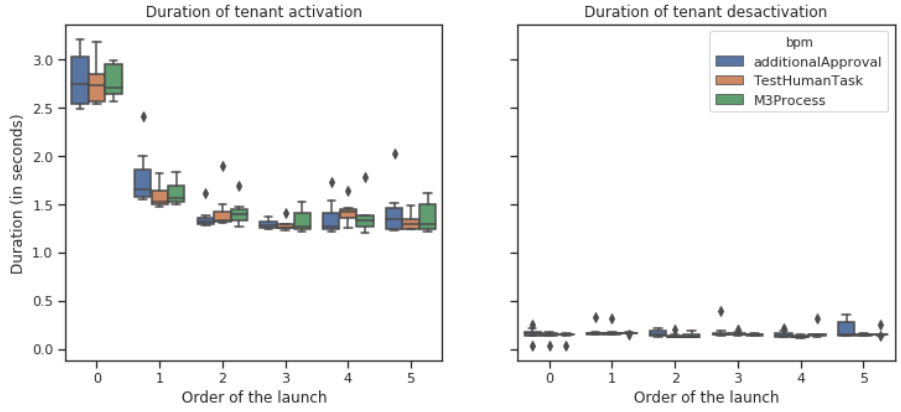

FIGURE 4.22 – Durée des phases d'activation et de désactivation en fonction de l'ordre de la migration

On peut y voir clairement que la durée est très stable pour la désactivation comme nous avons pu le constater sur la figure précédente. Elle est comprise entre 0,05 et 0,4 secondes pour l'ensemble des expérimentations. Ce n'est pas le cas pour l'activation correspondant à la première migration effectuée, qui a toujours une durée de plus de 3 secondes comparées aux 1,5 secondes des étapes suivantes. La deuxième migration a également une durée légèrement plus élevée que celles effectuées ensuite. Mis à part certaines migrations ponctuelles plus longues, le comportement de l'activation et de la désactivation est similaire quel que soit le schéma BPM utilisé, et la quantité de processus migrés.

Effets des migrations sur le tenant migré

Nous présentons dans le tableau 4.5 les statistiques générales des durées des processus avant (*pre*) et après (post) la migration du tenant 2. La durée moyenne des processus *TestHumanTask* est environ 1300 millisecondes plus courte après la migration comparée à avant, alors qu'elle est 1500 millisecondes plus longue pour le schéma *M3Process*, et 2700 millisecondes plus rapide pour le schéma *AdditionalApproval*. Les écarts-types sont similaires, hormis pour le processus *TestHumanTask*.

Schéma BPM	Moment	Quantité	Moyenne	Écart-type	Min	25%	50%	75%	Max
M3Process	post	10292,0	38449,298	16217,793	13657,0	26046,0	34401,0	44808,0	76321,0
M3Process	pre	14630,0	36950,857	15909,099	4398,0	24837,0	33933,5	47169,0	87303,0
TestHumanTask	post	19529,0	12579,335	9378,176	581,0	3081,0	9588,0	15921,0	32269,0
TestHumanTask	pre	19688,0	13892,846	15594,283	466,0	2969,5	12282,0	19553,5	168560,0
AdditionalApproval	post	17710,0	113370,094	42819,486	22915,0	78601,0	116355,0	146674,0	199938,0
AdditionalApproval	pre	25749,0	110685,854	42710,761	12907,0	83799,0	114749,0	141261,0	199856,0

TABLE 4.5 – Durée des processus groupés by schéma BPM et moment de la mesure.

Effets des migrations sur les tenants colocalisés

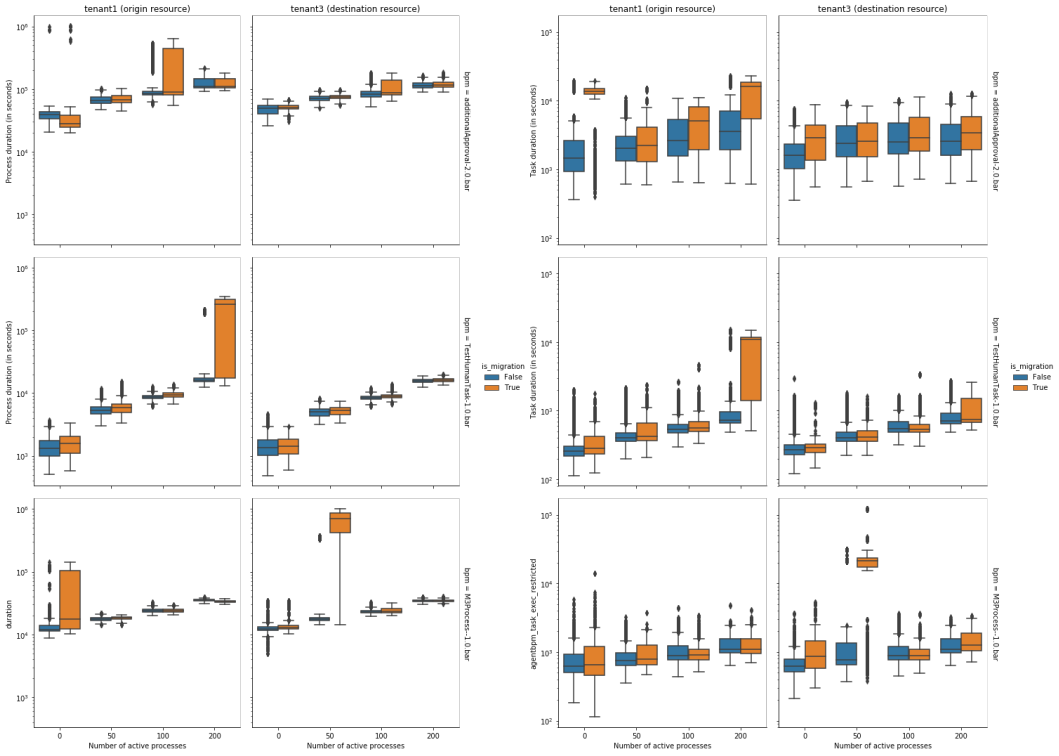

FIGURE 4.23 – Durée des processus selon moteur BPM (gauche) et durée des tâches selon agent BPM (droite)

On peut observer dans la figure 4.23 la durée des processus du point de vue du moteur BPM et celle des tâches du point de vue de l'agent BPM. Sur chaque graphique, la boite à moustaches bleue (à gauche) représente la durée quand la migration n'a pas lieu, et l'orange (à droite) représente la durée quand la migration a lieu.

On remarquera que beaucoup de variations ont lieu, probablement liées aux variabilités du Cloud. Celles-ci semblent avoir lieu plus souvent durant les migrations à chaud : Certains processus durent plus de 200 secondes, parfois jusqu'à 1000 secondes. Malgré ces effets, on peut constater des tendances : la durée des processus et tâches semble corrélée positivement avec le nombre de processus actifs. De plus, les durées sont généralement plus longues et moins stables durant les migrations, même si ce n'est pas le cas pour toutes les tests. Le même comportement est visible quand on observe les tâches du point de vue de l'agent BPM.

Afin d'avoir une comparaison homogène, nous avons décidé de retirer pour les étapes suivantes les processus dont la durée est supérieure à 200000 millisecondes, et les tâches de durée supérieure à 15000 secondes. Ces cas semblent ponctuels et ne pas être représentatifs de la plupart des

temps de réponse, comme on peut le constater dans les résultats. Le tableau 4.6 montre les résultats correspondants, groupés par tenant. Comme on peut le constater, la durée est toujours plus longue durant les migrations que ce soit pour le tenant colocalisé de la ressource d'origine (*tenant1*) ou celui de la ressource de destination (*tenant2*). L'écart de durée est compris entre 26 et 650 millisecondes en moyenne. Il ne semble pas y avoir de comportement spécifique à un schéma BPM, hormis les durées plus élevées du processus *AdditionalApproval* comparé à *M3Process* et *TestHumanTask*. Un comportement similaire survient pour la durée des tâches.

BPM schema	Tenant	Migr.	Qté	Moy.	Ecart-type	Min	25%	50%	75%	Max
M3Process	tenant1	False	11745,0	1018,967	511,425	183,0	668,0	884,0	1157,00	5819,0
M3Process	tenant1	True	4602,0	1126,497	695,887	115,0	738,0	962,0	1302,75	13926,0
M3Process	tenant3	False	11351,0	1003,217	475,908	208,0	674,0	874,0	1174,00	3612,0
M3Process	tenant3	True	3995,0	1208,9543	600,114	297,0	795,0	1025,0	1533,50	5270,0
TestHumanTask	tenant1	False	17375,0	718,012	1171,124	114,0	330,0	482,0	687,00	11671,0
TestHumanTask	tenant1	True	4107,0	734,108	464,686	124,0	439,0	617,0	791,50	4665,0
TestHumanTask	tenant3	False	18857,0	609,597	393,762	122,0	345,0	505,0	704,00	2959,0
TestHumanTask	tenant3	True	4667,0	738,623	491,750	145,0	426,5	603,0	791,00	3444,0
AdditionalApproval	tenant1	False	15131,0	2790,596	2055,715	360,0	1281,0	2100,0	3404,00	12241,0
AdditionalApproval	tenant1	True	5405,0	2883,677	2359,188	398,0	1313,0	2152,0	3484,00	14949,0
AdditionalApproval	tenant3	False	19530,0	3050,379	2109,850	354,0	1458,0	2255,5	4110,00	12673,0
AdditionalApproval	tenant3	True	8299,0	3702,043	2349,800	558,0	1751,0	2935,0	5521,00	12882,0

TABLE 4.6 – Durée des tâches pendant la migration comparée à avant, groupée par schéma BPM et tenant

4.3.7 Analyse des résultats

Nous présentons dans cette section notre analyse des résultats obtenus.

Durée des migrations

La durée d'une migration à chaud avec notre méthode varie entre 5 secondes pour aucun processus actif et 30 secondes pour le déplacement d'environ 10000 processus de type *TestHumanTask*. On remarque que les durées sont plus élevées pour des processus plus compliqués. On passe à environ 50 secondes pour 10000 processus de type *M3Process*, C'est probablement dû à la présence de deux tâches en parallèle, au lieu d'une seul pour les autres processus. La durée est comprise entre les deux pour le processus *AdditionalApproval*. Ceci doit être probablement provoqué par la présence de tâches automatiques et de branchements, qui provoquent la présence d'autres données actives que nous n'avions pas considérées. Cela montre que si l'on souhaite considérer les migrations à chaud, la structure des processus utilisés aura des effets sur la durée de migration.

Nous avons pu également constater que la durée des activations de tenants est plus longue pour les premières migrations. Pour rappel, nous avons considéré au total 6 migrations, 3 depuis le moteur BPM d'origine vers le moteur BPM de destination, et 3 depuis le moteur BPM de destination vers le BPM d'origine, sur le même tenant. L'explication probable est que la première activation d'un tenant dans une installation nécessite des initialisations de méta données dans les systèmes (telles que celles de bibliothèques, du cache, de ressources hébergées sur le système de fichier, etc.) qui n'existent pas encore au moment de l'activation en mémoire ou sur le système de fichiers. Ce comportement ne survient qu'à la première migration : celle-ci reste stable pour les migrations suivantes. Cette durée est donc à considérer lors de la première migration d'un tenant sur une ressource.

Effets des migrations sur le tenant migré

Comme nous l'avons précisé dans la partie précédente, nous n'avons pas eu de résultat probant à ce niveau, les temps moyens n'étant pas les mêmes selon le processus et l'intervalle considéré. Il est possible que ce comportement provienne de variations liées au Cloud [4]. Les durées semblent relativement similaires toutefois, des expérimentations plus intensives seraient nécessaires pour déterminer si le comportement provient de la structure, ou si on peut considérer qu'il n'y a pas d'effet de la migration.

Effets des migrations sur les tenants colocalisés

Les effets des migrations à chaud sur les tenants colocalisés ne sont pas très élevés dans les cas testés, mais existent clairement comme on pouvait s'y attendre. Quelques centaines de millisecondes supplémentaires sont à prévoir pour chaque type de schéma. On peut remarquer ces effets autant sur la configuration d'origine de la migration que la configuration de destination. Pour le processus *AdditionalApproval*, les migrations semblent avoir plus d'effets sur la configuration de destination. De plus précises investigations sont à prévoir, mais elles sont hors du cadre de cette thèse.

Limites de l'expérimentation

Notre expérimentation a quelques limites, que nous précisons ci-dessous :

— Nous avons testé les effets sur les tenants colocalisés et migrés pour de faibles quantité de processus actifs seulement (de 0 à 200).

— L'utilisation de ressources de cloud public pose des problèmes au niveau de la variabilité, en particulier chez les fournisseurs les plus utilisés tels que Amazon ou Azure [4]. Comme nous avons pu voir dans les résultats présentés, nous avons eu de nombreuses valeurs aberrantes. La solution serait de lancer plus d'expérimentations, et de vérifier les niveaux de confiance des résultats. Nous le ferons dans de futurs travaux.

— Les résultats ne concernent qu'un moteur BPM et une méthode de migration. Une généralisation des tests à d'autres solutions BPM permettrait de souligner un comportement des outils BPM multi-tenant et guiderait les utilisateurs pour leur choix, et les développeurs d'outil BPM pour améliorer les outils existants. Tester d'autres méthodes de migrations permettrait pour les fournisseurs de BPMaaS de choisir parmi celles-ci.

— Dans son implémentation actuelle, on peut faire passer le framework à l'échelle au niveau du nombre d'agents moteur BPM ou Faban. Cependant, quand nous avons tenté d'augmenter le nombre d'agents Faban, nous avons eu des problèmes de plantages au niveau du moteur BPM, probablement solvables avec un meilleur réglage de ce dernier. Dans cet expérimentation, nous avons réglé le moteur BPM de la même manière pour l'ensemble des expérimentations.

— Les résultats que nous avons obtenus avec notre framework d'évaluation de migrations restent à prouver statistiquement, à l'aide d'expérimentations plus poussées, mais nous donnent des indications générales remarquables concernant les effets des migrations.

— Nous n'avons pas étudié les effets de la fonctionnalité multi tenant sur les performances. Toutefois, nous ne sommes pas concernés par ce cas, vu que nous avons comparé nos résultats à chaque fois pour un nombre de tenant fixe. En effet, le temps de réponse pourrait ne pas être totalement proportionnel au nombre de processus injectés, dans le cas où il y aurait plus de tenants (comparé à une injection dans le même tenant). La durée des migrations est différente pour un nombre différent de tenant. De plus, la durée

de migration pourrait être différente selon le nombre de tenants colocalisés (ce qui s'avère être le cas).

4.3.8 Conclusion

Dans cette section, nous avons présenté trois contributions :

— une méthode de migration à chaud de type Stop-And-Copy pour applications Web multi-tenant,
— une approche générique servant à mesurer l'impact des migrations à chaud en termes d'interruption de service et d'évaluer des performances des applications Web,
— un framework d'analyse de performances pour moteur BPM basé sur cette approche.

Nous avons présenté comment le framework peut être utilisé pour évaluer un moteur BPM Open Source connu. Nos expérimentations nous ont donné des résultats intéressants sur les effets des migrations multi tenant à chaud sur la qualité de service. Les résultats montrent que la durée de la migration dépend du nombre de processus actifs et de la nature de ceux-ci. Nous avons pu également constater les effets des migrations sur les tenants colocalisés. Au vu de ces informations, nous pouvons en conclure que les effets des migrations à chaud de tenants sont à considérer pour les fournisseurs de BPMaaS, s'ils souhaitent fournir la même qualité de service pour leurs clients. La méthode que nous avons développée peut être utilisée pour cet effet, et notre framework utilisé pour analyser l'ensemble de ces effets, et ainsi prévoir

Notre travail nous a permis d'avoir une meilleure vision des effets des migrations à chaud multi-tenant et soulignent des critères à prendre en compte que les fournisseurs de Software as a Service ne devraient pas négliger. Les prochaines étapes à ce niveau concernent une étude plus intensive de l'effet des migrations sur les applications Web en général, à l'aide de plusieurs méthodes de migrations à chaud.

4.4 Conclusion du chapitre

Nous avons présenté dans ce chapitre les différentes hypothèses et métriques que nous employons dans nos algorithmes, et répondu à la question que nous avons posé dans la problématique sur le poids des migrations à chaud sur la qualité de service.

Pour ce point, nous avons développé une approche novatrice de migration à chaud pour applications transactionnelles multi-tenant, que nous avons utilisé pour nos estimations. Nous avons également proposé deux approches et framework, pour estimer la taille des ressources en regard de notre métrique de qualité d'un côté, et pour estimer l'effet des migrations à chaud sur les tenants migrés et colocalisés. Ces derniers sont fonctionnels et nous ont permis de justifier notre approche et d'utiliser des capacités et nombre de migrations ayant du sens. L'effet des migrations n'est pas négligeable pour les tenants colocalisés mais reste contrôlable. La durée de migration de son côté reste significative, vu qu'elle peut atteindre une minute d'indisponibilité. Il est important de la considérer dans les approches d'élasticité pour applications transactionnelles.

Une solution souvent utilisée en production est d'allouer chaque tenant à une ressource supportant son débit maximal. Celle-ci est simple et permet de s'affranchir des migrations à chaud, toutefois celle-ci peut s'avérer très chère. Nos méthodes d'optimisation permettent de réduire le coût des ressources employées et peuvent permettre aux fournisseurs de réaliser des économies, tout en contrôlant le nombre de migrations et en respectant les besoins des clients en termes de débit de nombre de tâches, comme nous le verrons dans le chapitre suivant.

Chapitre 5

Algorithmes d'allocation de ressources et de distribution de tenants

Dans ce chapitre, nous présentons les algorithmes que nous avons proposés pour répondre à la problématique d'élasticité de l'exécution des processus métiers dans le cloud.

Les hypothèses que nous avons utilisées pour nos algorithmes se résument ainsi :

Les BPMS ne passent pas à l'échelle pas de manière infinie Il n'existe pas d'installation de BPMS dont les performances augmentent de manière linéaire avec le nombre d'instances. Les installations en cluster atteignent une limite due à la naturelle transactionnelle des interactions avec les bases de données. Nous partons donc du principe que nous allons utiliser plusieurs installations BPM. Dans notre approche nous partons du principe que les capacités de la plus puissante des ressources peut héberger les besoins de chaque tenant.

L'allocation et la libération de ressources prennent du temps Il n'est pas possible de changer instantanément la distribution des tenants sur les instances de calcul, car la migration des données prend du temps et a des effets négatifs sur la qualité de service des tenants. Nous calculons donc l'allocation des ressources d'une manière discrète avec des intervalles temporelles fixes (ou pas de temps). Un pas de temps est une durée ayant du temps pour le fournisseur : elle peut être de quelques secondes, d'heure, etc.

Un tenant est un client d'un BPMaaS Les utilisateurs des tenants exécutent des processus BPM composés de tâches. Pour les exécuter, le moteur BPM nécessite de la puissance de calcul, de la bande passante réseau, de stockage, ainsi que de la mémoire. Elle s'appuie sur des instances de calcul séparées pour la base de données, le serveur d'applications ou les éventuels répartiteurs de charge (pour les installations en cluster).

Le débit de tâches est notre métrique de performance Le débit de tâche correspond au nombre de tâche maximal par seconde exécuté pour une durée fixée. Il correspond au nombre de tâches BPM exécutées par seconde pour une période de temps. Cette métrique a du sens pour les clients et extractible facilement depuis les données client pour les fournisseurs.

Notre approche est hors-ligne Nous partons du principe que nous connaissons à l'avance le débit de tâche requis pour chaque client et chaque pas de temps.

Une ressource du Cloud a une capacité exprimée en termes de débit de tâches BPM Une ressource du Cloud (ou *ressource*) est composée d'une ou plusieurs instances de calcul que nous utilisons pour la base de données, le serveur d'applications, les répartiteurs de charge, et est capable de supporter une installation complète de BPMS. Une ressource est capable d'héberger plusieurs tenants. Dans notre cas, nous partons du principe qu'à chaque moment, un tenant est capable d'être hébergé par une ressource unique : un tenant ne peut pas être hébergé sur plusieurs ressources.

Le nombre de migrations doit être limité Les migrations ont des effets sur les performances des tenants hébergés sur les ressources concernés, et provoquent des indisponibilités du tenant migré pouvant durer de quelques à une centaine de secondes (avec notre méthode de migration et jusqu'à 10000 processus). Nous choisissons de limiter leur nombre pour permettre un respect des contraintes de SLA.

Nous cherchons à déterminer pour chaque pas de temps une liste de ressources, ainsi que la distribution des tenants sur celles-ci.

Nous proposons dans ce chapitre trois algorithmes répondant à ce besoin :

— un premier modèle et une première heuristique dont le but est d'optimiser simultanément le coût et le nombre de migrations d'un pas de temps au suivant dans la section 5.1.
— une seconde heuristique basée sur la segmentation de séries temporelles, ainsi qu'un modèle généralisant ce principe à plusieurs pas de temps, pour un nombre de migrations fixe par tenant dans la section 5.2.
— une alternative basée sur les algorithmes génétiques dans la section 5.3. Nous y ajoutons une alternative à notre heuristique itérative présentée dans la section précédente.

5.1 Algorithme bi-objectif d'allocation de ressources et de placement des tenants pour un pas de temps

L'objectif de ce premier algorithme est de minimiser le coût des ressources de calcul et le nombre de migration de tenants, sous contrainte de débit de pas de temps. Comme nous avons pu voir dans la section 4.2.2 chapitre 4, les migrations ont des effets négatifs sur le temps de réponse et entraînent des périodes de non disponibilité au niveau du système. Cet algorithme est une première étape avant la généralisation à plusieurs pas de temps abordée dans les deux sections suivantes.

5.1.1 Modèle linéaire décrivant cet algorithme

Une installation est composée d'une ou plusieurs ressources du cloud, une pour la base de données, et une pour le serveur d'application par exemple. Plusieurs types de configurations, avec chacun un coût pour une durée donnée, et une capacité en termes de débit de tâches sont disponibles.

Soient les variables suivantes :

— $\mathcal{J}$, l'ensemble des configurations possibles avec m sa cardinalité
— C_j, et W_j, respectivement le coût et la capacité de la configuration j
— $\mathcal{I}$, l'ensemble des tenants avec n sa cardinalité
— w_i, les besoins en capacité du tenant i durant le pas de temps $k+1$
— ${x_j}^i(k)$, l'assignation du tenant i à la configuration j durant le pas de temps k
— $y_j(k)$, l'activation de la configuration j durant le pas de temps $k+1$

Nous définissons une fonction indicatrice $\mathbb{1}_{\{x_j{}^i(k) \neq x_j{}^i(k+1)\}}$. qui correspond à la migration d'un tenant. Celle-ci sera égale à : $\begin{cases} 0 \text{ si } x_j{}^i(k) = x_j{}^i(k+1) \\ 1 \text{ si } x_j{}^i(k) \neq x_j{}^i(k+1) \end{cases}$

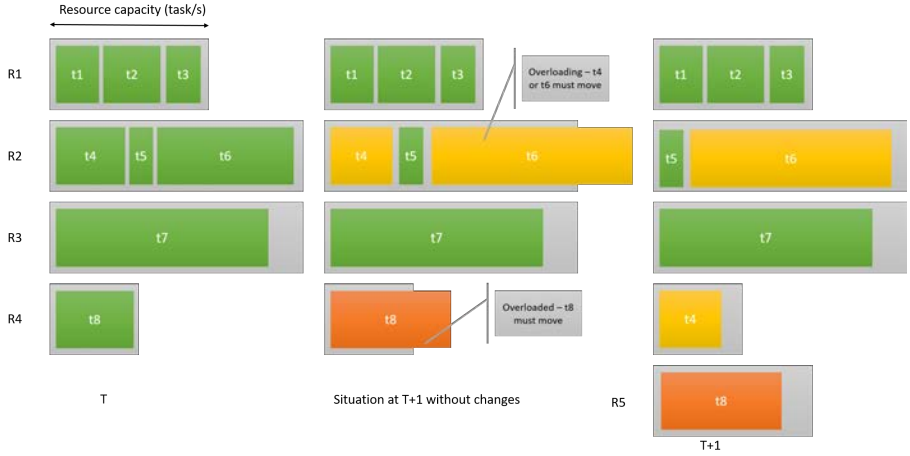

FIGURE 5.1 – Exemple de distribution de tenants t1 à t8 sur des ressources R1 à R5 au pas de temps T, avec la charge du pas de temps T (gauche), avec la charge du pas de temps T+1 (milieu), et distribution valide au pas de temps T+1 (droite).

Le but est de minimiser le coût total des *configurations actives* et le nombre de migrations entre le pas de temps k et $k+1$. On commence par appliquer aux distributions de configuration du pas de temps k les besoins de capacité du pas de temps $k+1$. Une *configuration active* est une configuration activée durant un pas de temps donné. La figure 5.1 présente ceci avec 8 tenants distribués entre 4, puis 5 ressources. L'évolution de la charge de certains tenants entre les pas de temps T et T+1 provoque une situation où les demandes des clients ne sont pas respectées. La couleur des tenants indique si un tenant a un débit trop élevé pour sa ressource (rouge), s'il n'y a pas suffisamment d'espace pour tout héberger simultanément (jaune), ou s'il y en a suffisamment (vert). Les ressources ayant une capacité définie et fixe, déplacer les tenants s'avère ici nécessaire.

Le problème peut être représenté sous la forme du modèle d'optimisation linéaire suivant. On souhaite optimiser les fonctions objectif suivantes :

$$\min f1 = \sum_{j}^{j \in \mathcal{J}} C_j y_j(k+1) \tag{5.1}$$

$$\min f2 = \sum_{j}^{j \in \mathcal{J}} \sum_{i}^{i \in \mathcal{I}} \mathbb{1}_{\{x_j{}^i(k) \neq x_j{}^i(k+1)\}} x_j{}^i(k+1) \tag{5.2}$$

Sur lesquelles nous avons les contraintes suivantes :

$$\forall i \in \mathcal{I} \sum_{j}^{j \in \mathcal{J}} x_j{}^i(k+1) = 1 \tag{5.3}$$

$$\forall j \in \mathcal{J} \sum_{i}^{i \in \mathcal{I}} w_i x_j{}^i(k+1) \leq W_j y_j(k+1) \tag{5.4}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}, {x_i}^j \in \{0,1\}, y_j \in \{0,1\} \tag{5.5}$$

L'équation 5.5 représente les variables en présence telles que décrites plus haut. Celles-ci sont binaires : un tenant est assigné ou non à une ressource, et une ressource est active ou non. 0 correspond à une non assignation ou une ressource désactivée, et 1 à une assignation ou une ressource activée, selon la variable considérée (x ou y).

L'équation 5.1 représente la minimisation du coût complet de l'ensemble des ressources multiplié par leur activation au pas du temps $k+1$. Le moins cher serait évidemment de n'avoir aucune ressource active au pas de temps $k+1$.

L'équation 5.2 représente la somme des indicatrices décrites plus haut pour une assignation tenant-ressource différente entre les deux pas de temps. Le minimum revient à n'effectuer aucune migration. Les équations suivantes permettent ensuite d'assurer une distribution cohérente des tenants.

L'équation 5.3 signifie que, pour chaque tenant, la somme des assignations à chaque ressource doit être égale à 1. Un tenant doit être forcément affecté à une (et une seule) ressource au pas de temps $k+1$.

L'équation 5.4 représente la capacité des ressources à ne pas dépasser. Pour chaque ressource, la somme des besoins des tenants affectés à cette même ressource doit être en dessous de la capacité de cette ressource.

Pour un nombre défini de ressources, et sans prendre en compte les migrations, il s'agit d'un problème classique d'assignation. Ces contraintes supplémentaires rendent le problème plus compliqué à résoudre. De plus, comme nous le préciserons dans la section suivante, le problème est NP-difficile. Il est approprié ici de proposer des heuristiques, la complexité du problème étant exponentielle.

5.1.2 Heuristique proposée

L'approche que nous proposons couvre deux parties : l'allocation des ressources du cloud et l'assignation des tenants à ces ressources. Ces deux types de problèmes sont NP-difficiles. Les algorithmes gloutons, la programmation linéaire, ou la programmation par contraintes sont habituellement utilisées pour les résoudre [109].

Les algorithmes gloutons sont des heuristiques qui partent du principe qu'en visant la meilleure solution locale, une bonne solution globale peut être atteinte. Ce type d'algorithme est habituellement simple et rapide, mais peut rester bloqué sur un optimum local. La programmation linéaire en nombre entiers (Integer Linear Programming - ILP) et la programmation linéaire partiellement en nombres entiers (Mixed Integer Linear Programming - MILP) sont des algorithmes de recherche opérationnelle cherchant à minimiser une fonction objectif sous contraintes à l'aide de variables discrètes (ou un mélange de variables discrètes et continues dans le cas d'un MILP). La résolution de ce type de problème passe habituellement par une relaxation du caractère discret des variables présentes. Ceci est bien plus rapide à résoudre et donne une borne inférieure (pour un problème de minimisation) au problème. Des étapes supplémentaires permettent d'obtenir la solution réelle. Toutefois, selon la taille du problème le temps de calcul peut devenir extrêmement long.

Nous choisissons de proposer une heuristique simple, pouvant calculer une solution proche de l'optimum en un temps bien plus court qu'à l'aide d'un solveur, comme nous le verrons dans les résultats de l'expérimentation.

Rappelons que notre but ici est de trouver les résultats optimums pour le coût et le nombre de migrations simultanément. Nous avons choisi le critère de Pareto pour répondre à ce besoin.

Le caractère multi-objectif de notre problème rend cela plus complexe à résoudre. Pour pallier à ce problème, nous avons choisi de profiter du caractère discret et limité du nombre de migrations. En effet, ce dernier sera forcément compris entre zéro et le nombre de tenants considéré. Ensuite, pour chaque nombre de migrations, notre principe est de trouver la solution la moins chère. Réduire le coût signifie réduire le nombre et la taille en termes de coût des configurations actives tout en respectant les contraintes. Cette considération nous a amenés à rapprocher ce nombre de migrations des ressources actives. Pour chaque nombre de migrations, on peut considérer un ensemble de ressources, dont le nombre de tenants cumulés correspond au nombre de migrations. Bien sur, il peut exister plusieurs combinaisons de ressources amenant à ces nombres de tenants.

Nombre de migrations

Le premier axe de notre approche est de considérer pour chaque nombre de migrations les combinaisons de ressources dont le nombre de tenants correspond au nombre étudié. Il s'agit d'un problème de somme de sous-ensembles. Le problème de la *somme de sous-ensembles* (*Subset Sum Problem*) est un problème d'optimisation très connu [110]. Le but est de retrouver pour un ensemble de nombres entiers donnés la combinaison dont la somme correspond à une valeur attendue. Il s'agit d'une variation du problème du sac-à-dos [111]. Voici une définition formelle.

Soit un ensemble N de n éléments possédant des poids entiers positifs w_i avec $i \in \{0, n\}$ et une capacité c. Le but du problème de somme de sous-ensembles est de trouver un sous-ensemble de N tel que le poids total correspondant est maximisé sans dépasser la capacité c.

Dans notre approche, nous avons décidé de ne pas considérer l'échange de tenants, et de nous concentrer sur la réduction (ou l'ajout) des ressources. Toutefois, dans notre cas, il ne nous est pas nécessaire de prêter attention à l'ensemble des nombres de migrations dans la somme de sous-ensembles. En effet, on peut considérer par exemple que les tenants dont les besoins excèdent la capacité de leur ressource sont forcément à déplacer. Ceci nous a amené à réfléchir à ces cas de figure.

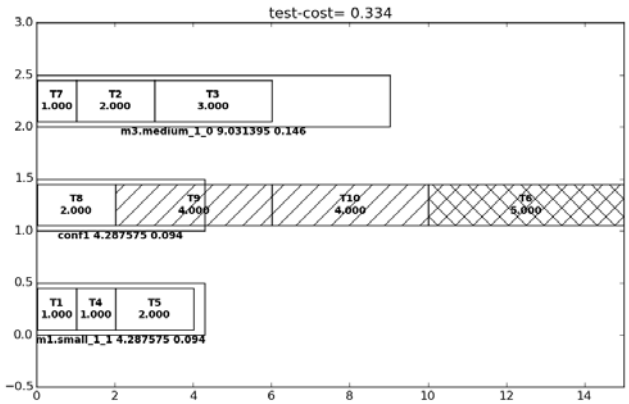

FIGURE 5.2 – Exemple de distribution de configurations et de tenants pour un pas de temps avec la charge des pas de temps suivants.

Quand on observe les situations de surcharge possibles, on peut discerner les deux cas suivants, illustrés par la figure 5.2. Dans cette figure, trois ressources contiennent dix tenants - nommés T1 à T10 ici. Les tenants à rayure représentent les tenants dont les futurs besoins cumulés dépassent

la capacité de leur ressource. Le tenant à double rayure aura une charge supérieure à celle de sa ressource, et devra être forcément migré sur une autre ressource.

— les tenants qu'il est nécessaire de migrer parce que leur futur besoin est plus grand que la capacité de leur ressource actuelle. La migration de ce type de tenants est obligatoire. Le tenant T6 de la figure 5.2 en est un exemple.

— en dehors de cette première catégorie, les tenants qui, placés ensembles, surchargeraient leur ressource actuelle au pas de temps suivant. En considérant les tenants par besoins décroissants, ceux qui cumulés aux autres auraient des besoins cumulés à leur ressource. Dans la figure 5.2 au niveau de la deuxième ressource, on considère les tenants T10, T9 et T8. Dans cet exemple, pour que la capacité de la ressource ne soit pas dépassée, il faudrait retirer T10 et T9.

Pour les étapes suivantes, on retiendra les définitions suivantes :

Definition 7. *Une ressource est dite* ***surchargée*** *si la somme des besoins futurs des tenants qu'elle héberge dépasse sa capacité.*

Definition 8. *Un tenant est dit* ***inadapté*** *si ses besoins futurs dépassent la capacité de sa ressource.*

Definition 9. *Un tenant est dit* ***surchargeant*** *s'il n'est pas inadapté et si ses besoins cumulés aux autres tenants provoquent une surcharge de la ressource, en considérant les besoins des tenants de la ressource de manière décroissante.*

Definition 10. *On nomme* ***distribution*** *l'ensemble des ressources actives et l'assignation des tenants à ces ressources pour un pas de temps de donné.*

Notre but est de trouver les *distributions* les moins chères pour chaque nombre de migration.

Le nombre minimal de migrations à effectuer correspond au nombre total de tenants *inadaptés* et *surchargeants*. En effet, la considération des tenants *surchargeants* de manière décroissante permet de sélectionner en premier lieu les tenants dont les besoins sont les plus élevés, et de les retirer de la ressource surchargée en premier lieu. Une fois ceux-ci considérés, nous nous concentrons sur le nombre de migrations, par le biais des ressources.

Selon ces principes, dans la figure 5.2, le nombre minimal de migrations est de 3, le tenant T6 étant *surchargeant*, et les tenants *T9* et *T10 inadaptés*. Ces tenants seront forcément migrés. Pour 4 migrations, nous avons seulement la possibilité de considérer la ressource *conf1*, avec le tenant T8 (les autres tenants de cette ressource étant déjà comptés). Il n'y a pas de solution pour 5 migrations, et deux possibilités pour 6 migrations : considérer la ressource *m3.medium_1_0* (avec les tenants *T7*, *T2*, et *T3*), *ou* la ressource *m1.small_1_1* (avec les tenants *T1*, *T4*, et *T5*).

Il faut retenir ici que le nombre de possibilités augmente de manière exponentielle avec le nombre de ressources et de tenants. Sans limitations, ceci amènera des problèmes de mémoire et de temps de calcul. Imaginons par exemple que nous ayons 50 tenants, tous hébergés sur une configuration différente. Dans le cas où nous utiliserions cet algorithme de manière non limitée, nous obtiendrions pour 25 migrations C_{25}^{50} combinaisons, ce qui signifie plus de 10^{14} combinaisons à évaluer. Nous avons donc ajouté une limite au nombre de combinaisons étudié pour chaque nombre de migrations. Comme nous le verrons dans la section concernant l'expérimentation, nous avons choisi de tester au maximum 1000 possibilités.

Allocation de ressources et assignation des combinaisons

Pour chaque nombre de migrations, nous nous appliquons ensuite à trouver la combinaison de ressources ayant le coût le plus faible. Notre principe d'affectation pour chacune des combinaisons étudiées est de considérer que les ressources sont virtuellement supprimées, laissant ainsi des tenants sans ressources. Nous nommons ces tenants en attente de placement, tenants **orphelins**. Ces tenants doivent avoir une ressource les hébergeant. Pour cela, nous avons les solutions suivantes :

— nous les réaffectons à des ressources ayant suffisamment de capacité disponible, il s'agit ici de *repacking*. Avec la réutilisation de ressources existantes, le coût complet reste identique.

— nous les affectons à de nouvelles ressources. Il s'agit ici de *bin packing*. De nouvelles ressources, inexistantes dans le pas de temps précédent sont créées dans le but d'héberger les tenants *orphelins*. Celles-ci doivent évidemment être les moins chères possibles.

Nous avons pris comme principe de réaliser sur l'ensemble des tenants *orphelins* une étape de *repacking*, puis, si les tenants n'ont pas pu être redistribués sur les ressources existantes, une étape de *bin packing* sur les tenants *orphelins* restants. Vu que l'ajout de nouvelles ressources fournit de nouvelles possibilités pour le placement des ressources, il y a une possibilité de trouver des affectations moins chères. En effet, les ressources ajoutées n'étant pas forcément totalement remplies, il est envisageable de placer des tenants de grande taille utilisés dans la phase de repacking. Ceci peut amener à replacer d'autres tenants de la deuxième phase et éventuellement obtenir des gains en termes de création de nouvelles ressources. Nous avons ajouté ici une étape de *consolidation* dans le but de répondre à cette considération.

Algorithm 1 Boucle principale

```
1:  procedure BOUCLE PRINCIPALE(distribution des tenants pour l'heure précédente, besoin des tenants pour l'heure suivante)
2:      resultNumberLimit = 1000
3:      unfit ← tenants whose capacity is bigger than their current ressource
4:      overloading ← biggest tenants to remove until there is no more overloaded resource
5:      mandatoryTenants ← unfit ∪ overloading
6:      minMigrations = size(mandatoryTenants)
7:      maxMigrations = size(tenants)
8:      distribution = remove mandatoryTenants from distribution
9:      for i = minMigrations → maxMigrations do
10:         costByMigration[i] ← +∞
11:         possibleResourcesCombinations ← FINDSUBSETSUM(i, resources)
12:         for all resourceCombination ∈ possibleResourcesCombinations do
13:             newDistribution ← distribution - resources ∈ resourceCombination
14:             tenantsToRepack ← mandatoryTenants tenants ∈ resources ∈ resourceCombination
15:             REPACKING(newDistribution, tenantsToRepack)
16:             BINPACKING(newDistribution, tenantsToRepack)
17:             CONSOLIDATION(newDistribution, tenantsToRepack)
18:             if cost(newDistribution) ≤ costByMigration[i] then
19:                 costByMigration[i] ← cost(newDistribution)
20:                 distributionResult[i] ← newDistribution
21:     return distributionResult
```

L'ensemble de ces étapes est décrit dans l'algorithme 1. Celui-ci reprend l'ensemble des points que nous avons décrits. Nous commençons par déterminer la liste des tenants *inadaptés* (ligne 3), en comparant pour chaque tenant si ses besoins sont supérieurs à la capacité de leur ressource actuelle. Nous obtenons le tableau unfit. La deuxième étape consiste à considérer les tenants non

inadaptés par ordre de besoins décroissant pour chaque ressource. Nous considérons la somme des tenants pouvant rester sur la ressource, puis retirons les besoins de chaque tenant dans l'ordre jusqu'à ce que la somme des besoins des tenants restants n'excède plus la capacité. Les tenants retirés sont stockés dans le tableau *overloading* (ligne 4). Le tableau *mandatoryTenants* contiendra l'ensemble des tenants *inadaptés* et *surchargeants* (ligne 5). Dans la ligne 8, nous considérons une distribution où nous avons retiré ces derniers.

Nous allons décrire maintenant chacune des fonctions.

L'étape de récupération des sommes de sous-ensembles prend en paramètre le nombre de migration, la distribution concernée. Notre principe ici est d'observer l'ensemble des combinaisons comme le montre l'algorithme 2. Nous avons adapté le code de la version "brute force" de l'algorithme décrit dans [112].

Algorithm 2 Somme de sous-ensembles

```
1:  procedure SUBSET SUM(migrationsByConfiguration, migrationNumber)
2:      answer = ∅
3:      for i = 1 → len(migrationsByConfiguration) − 1 do
4:          for all comb ∈ combinations(migrationsByConfiguration, i) do
5:              total ← 0
6:              for all c ∈ comb do
7:                  total ← total + c
8:              if total = migrationNumber then
9:                  answer ← answer + comb
10:     return answer
```

Le principe général est de rechercher de manière exhaustive l'ensemble de toutes les combinaisons de tenants contenant le nombre de migrations. Pour cela nous observons itérativement pour tous les nombres de tenants (ligne 1), les combinaisons possibles. Ceci est réalisé en ligne 4 grâce la méthode *combinations*. Pour chacune de ces combinaisons, nous retenons celles dont la somme correspond au nombre de migrations et la rajoutons ensuite au tableau *answer*, en ligne 8. Celui-ci est ensuite renvoyé. Même si l'ensemble des combinaisons correspondantes est retourné, ne seront traitées dans les phases suivantes que le nombre maximal de combinaisons considérées.

L'étape de "repacking" consiste à replacer les tenants orphelins. Pour cela nous nous sommes appuyées sur un algorithme de type "Best Fit Decreasing", tout en considérant les capacités restantes des ressources de manière décroissante. En effet, on peut considérer l'ensemble des ressources comme autant de boîtes de taille fixe. Chaque taille correspondant à la capacité de la ressource à laquelle on retire la somme des tenants restants sur celle-ci. Nous ne prenons pas en compte ici les tenants *inadaptés* et *surchargeants*. Nous le décrivons dans l'algorithme 3.

Algorithm 3 Algorithm de repacking

```
1: procedure REPACKING(distribution, orphanTenants)
2:     sortedResources ← resources from distribution sorted by remaining loads decreasingly
3:     sortedTenants ← orphanTenants sorted by needed throughput decreasingly
4:     for all resource ∈ sortedResources do
5:         for all tenant ∈ sortedTenants do
6:             if size(tenant) < remainingLoad(resource) then
7:                 orphanTenants ← orphanTenants - tenant
8:                 add tenant to resource
```

L'étape de bin packing doit prendre en compte le coût et la taille variable des ressources.

Un algorithme simple de bin packing, s'appliquant à minimiser un nombre de bins identiques ne suffit pas dans notre cas de figure. Ce type de problème est un sujet actif de recherche, et est nommé VCSBPP (Variable Cost and Size Bin Packing Problem) [113] mais plus souvent VSBPP (Variable Size Bin Packing Problem) [114], [115], [116]. Pour l'étape de bin packing, nous avons employé l'algorithme Iterative Best Fit Decreasing de Kang et al. [117] qui reste une référence en la matière et propose une heuristique simple. Nous la décrivons dans l'algorithme 4. Le principe général ici d'un côté est de considérer les types de ressources disponibles les meilleurs marchés en premier, et les tenants par taille décroissante comme dans l'approche Best Fit Decreasing de l'autre.

Algorithm 4 Algorithme de variable size bin packing

```
1: procedure BINPACKING(distribution, orphanTenants, resourceTypes)                    ▷
2:     sortedTypes = resourceTypes sorted by cost by task
3:     sortedTenants = orphanTenants sorted by needed throughput decreasingly
4:     while len(orphanTenants) ≥ 0 do
5:         currentTenants ← ∅
6:         for all type ∈ sortedTypes do
7:             for all tenant ∈ sortedTenants do
8:                 currentSumTenantLoad ← ∑ neededLoad(tenants ∈ currentTenants)
9:                 if currentSumTenantLoad > theoricLoad(type) then
10:                    break
11:                else
12:                    selectedType ← type
13:                    currentTenants ← currentTenants ∪ tenant
14:                    tenants ← tenants − tenant
15:            if len(currentTenants > 0 then
16:                break
17:            for all tenant ∈ configurationTypesSorted do
18:                if ∑ neededLoad(tenants ∈ currentTenants) ≤ theoricLoad(type) and
   len(currentTenants)> 0 then
19:                    if price(type) < price(selectedType) then
20:                        selectedType ← type
21:            if selectedType = ∅ and (len(currentTenants) > 0) then return ERROR
22:            configuration ← new Configuration(selectedType)
23:            for all tenant ∈ currentTenants do
24:                configuration.tenants ← configuration.tenants ∪ tenant
25:            distribution ← distribution ∪ configuration
```

L'étape de consolidation - décrite dans l'algorithme 5 - s'applique à tenter de profiter de l'espace restant des ressources des deux précédentes étapes. Nous avons pris comme principe ici de supprimer virtuellement les ressources une par une, puis de tenter de replacer les tenants orphelins résultants dans les ressources restantes. Cette étape est inspirée par un des opérateurs de la recherche local utilisée par Hemmelmayr et al. [116]. Nous l'avons adapté à nos besoins, en prenant en compte uniquement les ressources qui contiennent initialement des tenants *orphelins*. Nous avons réutilisé notre algorithme de repacking ici pour chaque ressource supprimée.

Une fois l'ensemble de ces étapes déclenchées, nous obtenons une *distribution*. Ces trois étapes sont lancées sur chaque combinaison fournie par la somme de sous-ensemble pour le nombre de migrations considéré. Pour un nombre de migration donné, nous retenons la *distribution* la moins chère parmi toutes celles évaluées. Ceci nous permet de constituer le front de Pareto correspondant à notre problème. Un exemple est présenté dans la figure 5.3. Les gros points représentent

Algorithm 5 Algorithme de consolidation de ressource

```
1: procedure CONSOLIDATION(distribution, orphanTenants)                    ▷
2:     sortedResources = resources from distribution sorted by loads increasingly
3:     for all resource ∈ sortedResources do
4:         if l thenen(resource) > 0
5:             if (∀tenant ∈ resource) ∈ orphanTenants then
6:                 newDistribution ← distribution − resource
7:                 removeTenants ← tenants ∈ resource
8:                 REPACKING(distribution, removeTenants)
9:                 if len(removeTenants) = 0 then
10:                    distribution ← newDistribution
```

ceux de la frontière de Pareto. L'absence de point pour un nombre de migrations donné signifie qu'il n'y a pas de combinaison de ressources correspondant à ce nombre de migrations. Le front de Pareto comprend trois solutions, respectivement pour 3, 4 et 7 migrations. Les solutions pour 6, 9 et 10 migrations n'en font pas partie, leur coût étant supérieur ou égal à des coût pour des nombres de migrations inférieurs.

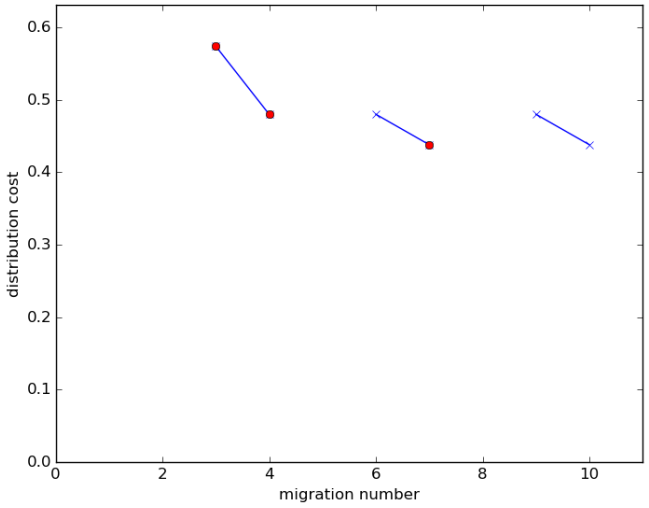

FIGURE 5.3 – Représentation des meilleurs résultats de notre algorithme après un lancement sur la distribution de la figure 5.2.

5.1.3 Multi-objectif et implémentation du modèle

Afin de pouvoir comparer modèle et heuristique, nous avons implémenté le modèle dans un solveur durant notre expérimentation. Toutefois, le caractère multi-objectif de notre modèle le rend difficile à résoudre avec les solveurs d'optimisation linéaires commerciaux standards tels que Cplex ou Gurobi. Nous avons tenu à rester sur des algorithmes et solutions simples, et préféré adapter le modèle au caractère mono-objectif de ce type de solveur. En l'occurrence, notre approche basée sur le caractère discret de l'objectif "nombre de migrations" rend possible l'optimisation de manière simple. Ici, nous avons transformé la fonction objectif de l'équation 5.2 en contrainte, présentée dans l'équation 5.6.

$$\sum_{j}^{j\in\mathcal{J}}\sum_{i}^{i\in\mathcal{I}}\mathbb{1}_{\{{x_j}^i(k)\neq{x_j}^i(k+1)\}}\,{x_j}^i(k+1)=M \tag{5.6}$$

Pour chaque ressource, et la somme des indicatrices -représentant une migration- multipliées par la variable d'assignation de chaque tenant à cette ressource doit être égale au nombre de migrations considéré. La variable M représente ce nombre de migrations. Le principe est de résoudre ce modèle pour chaque nombre de migrations (en comptant le nombre de migrations minimal fourni par la quantité de tenants *inadaptés* et *surchargeants*). Nous obtiendrons ainsi le front de Pareto. La résolution de ce modèle remplace les phases de récupération de la somme de sous-ensembles, de repacking, de bin packing et de consolidation de l'algorithme 1, comme on peut le voir dans l'algorithme 6.

Algorithm 6 Boucle principale avec solveur

```
1: procedure BOUCLE PRINCIPALE AVEC SOLVEUR(distribution des tenants pour l'heure précédente, besoin des tenants pour l'heure suivante)
2:     unfit ← tenants ina
3:     overloading ← biggest tenants to remove until there is no more overloaded resource
4:     mandatoryTenants ← inadaptes ∪ surchargeants
5:     minMigrations = size(mandatoryTenants)
6:     maxMigrations = size(tenants)
7:     distribution = remove mandatoryTenants from distribution
8:     for i = minMigrations → maxMigrations do
9:         newDistribution ← solver(previous distribution, needed load, resources cost and capacity)
10:        costByMigration[i] ← cost(newDistribution)
11:    return distributionResult
```

Le solveur prend en paramètre la distribution des tenants de l'heure précédente, les besoins des tenants pour l'heure suivante, ainsi que les capacités et prix des ressources. Il renvoie une distribution. Le reste de l'algorithme fonctionne de la même manière.

5.1.4 Expérimentation

Afin d'évaluer la qualité de notre algorithme, nous avons comparé pour chaque nombre de migrations les résultats de notre algorithme avec ceux d'un solveur - pour un temps limité.

Capacités des ressources

Nous avons besoin en premier d'une estimation de la capacité de chaque type de ressource. Pour cela, nous nous sommes appuyés sur une version plus ancienne du framework décrit dans la section 4.2.2 du chapitre 4. Concernant la partie logicielle, nous nous sommes appuyés sur la version 7.0.3 de la version open-source de BonitaBPM, et sur la version 9.3 de la base de données Postgresql. L'injecteur de processus est le même que celui présenté dans le chapitre 4, la seule différence étant que cette version a été compilée avec les références de la version 7.0.3 du moteur BPM. Le processus reste également le même. Nous avons testé ici les capacités d'instances EC2 à usage générique de la famille *m3* pour l'instance de la base de données et l'instance du serveur d'application et du BPMS.

Les résultats obtenus sont précisés dans le tableau 5.1. La configuration le meilleur marché à employer (c'est à dire ayant le débit le plus élevé pour un dollar) est la m3.medium/m3.medium.

Type instance DB	Type instance BPMS	Prix	Débit tâche	Débit tâche par $
m1.small	m1.small	0,094	8,120	86,392
m3.medium	m3.medium	0,146	17,762	121,660
m3.medium	m3.large	0,219	24,669	112,644
m3.medium	m3.xlarge	0,366	25,293	69,107
m3.large	m3.xlarge	0,439	41,147	93,730
m3.large	m3.2xlarge	0,693	42,813	61,868
m3.xlarge	m3.2xlarge	0,839	45,274	53,962

TABLE 5.1 – Prix, débit de tâches moyen, et débit de tâches moyen par dollar pour un processus standard d'une durée de 10 secondes

Besoins des tenants

Nous devons avoir également pour chaque tenant une mesure des besoins de celui-ci en termes de débit de tâches BPM. A ce propos, nous nous sommes appuyés sur des seuils tirés d'historiques de clients anonymes de Bonita BPM. Nous avons observé la quantité totale par tâches exécutées par seconde pour six d'entre eux. Nous avons gardé les quantités maximum et minimum, telles que décrites dans le tableau 5.2.

Client	Durée observée	Minimum	Maximum
A	4	2	45
B	1	14	16
C	45	0	45
D	7	1	3
E	45	5	45
F	550	0	4

TABLE 5.2 – Synthèse des données utilisées, faisant figurer le client, la durée observée, ainsi que le minimum et le maximum des besoins du client en termes de débit de tâches par seconde

Pour déterminer les besoins des tenants de l'expérimentation, nous nous sommes appuyés sur ces seuils, en prenant des valeurs au hasard comprises dans cet intervalle, ainsi que des clients au hasard parmi les 6 retenus.

Considérations techniques

Nous avons développé l'ensemble des algorithmes en Python. Le modèle a été résolu par le biais du solver Gurobi [41], grâce à la version académique gratuite. Afin d'interfacer notre code, nous avons utilisé la bibliothèque PuLP [42] dont le but est de fournir une interface simplifiée aux pour les problèmes d'optimisation linéaire, et est intégrable avec les principaux solveurs du marché.

Nous avons effectué nos calculs sur une instance Amazon Web Services EC2 de type *c4.xlarge*. Les instances de la famille *c4* sont optimisées pour le calcul intensif. Nos calculs ont été lancés sur notre heuristique, et sur une résolution du modèle à l'aide des outils décrits. Il est à retenir que nous avons posé une limite de 3 heures aux calculs du solveur. Pour l'heuristique, nous avons lancé 30 fois l'algorithme avec des distributions différentes, pour plusieurs nombres de tenants.

41. `http://www.gurobi.com/`
42. `https://pythonhosted.org/PuLP/`

Afin de considérer des distributions représentatives, nous avons à chaque fois exécuté deux fois l'algorithme sur une configuration initiale où chaque tenant était hébergé sur la configuration la plus petite, sans considérer la charge du tenant. Nous avons alors lancé deux fois l'heuristique. Une première fois pour obtenir la configuration initiale, puis une deuxième pour comparer les résultats de notre algorithme. Afin de pouvoir comparer de manière effective les résultats, nous avons lancé la deuxième itération sur les mêmes résultats, pour l'heuristique et pour la résolution du solveur.

5.1.5 Résultats obtenus et analyse

Quantité de tenants (nombre)	Résultats solveur (nombre)	Durée heuristique (secondes)	Durée solveur (secondes)	Pourcentage front Pareto %	Efficacité algorithme %
5	30	0,003	0,128	85,55	1,41
10	30	0,047	0,864	85,38	0,97
20	30	5,243	13,625	81,55	1,97
30	29	78,56	579,31	78,56	2,26
40	29	1013,85	1850.66	66,22	4,10

Table 5.3 – Résultats de l'expérimentation

Les résultats obtenus sont décrits dans le tableau 5.3. Sur les 30 expérimentations, deux cas n'a pas donné de résultat par solveur au bout des 3 heures de calcul. Nous avons choisi ici de ne pas considérer le résultat obtenu par le solveur, celui-ci n'étant pas optimal. Le pourcentage du front de Pareto correspond au ratio d'optimums par migration trouvés dans l'heuristique vis-à-vis des optimums par migration du solveur. Pour l'ensemble des résultats faisant partie du front de Pareto dans les deux cas, nous avons calculé l'efficacité de l'heuristique en calculant sa différence avec le résultat exact, divisé par le résultat exact.

On peut remarquer ici que les résultats obtenus par l'heuristique sont très intéressants, avec une distance à la solution n'excédant pas 4.1%. De plus, dans le pire cas étudié, le pourcentage du front de Pareto n'excède pas 66.22%, et l'heuristique reste plus rapide que l'algorithme. Les résultats que nous avons obtenu montrent que, tout en durant une fraction du temps employé par le solveur, l'heuristique donne de bons résultats avec des erreurs faibles, pour la plupart des nombres de migrations faisant partie du front de Pareto optimal. Même si le pourcentage du front de Pareto baisse avec le nombre de tenants, on peut réaliser ici que l'efficacité de l'heuristique reste très élevée. Ce dernier point montre que les deux fronts de Pareto (exacts et approchés) restent très proches.

Malgré l'utilisation d'une approche simple pour l'évaluation des sommes de sous-ensembles ne considérant que 1000 combinaisons, nous obtenons de très bons résultats. Cette étape prend la plupart du temps de calcul pour les grands nombres de tenants. Utiliser une version approchée de l'algorithme en lieu et place de la version récursive devrait accélérer notre algorithme. Une autre amélioration possible est l'utilisation d'une version multi-threadée, au lieu de la mono-threadée que nous avons utilisée. Ceci permettrait de profiter pleinement de l'architecture multi-coeur des processeurs actuels, ce que fait le solveur Gurobi pour ses calculs. Par exemple, plusieurs combinaisons pourraient être calculées simultanément comme nous le verrons dans l'évolution de cet algorithme.

5.1.6 Conclusion

Nous avons présenté dans cette section une approche efficace pour l'exécution des processus métiers dans le cloud. Celle-ci considère les hypothèses décrites dans le chapitre précédent. Elle s'applique à minimiser deux objectifs conflictuels : le coût et le nombre de migrations, et fournit comme résultat la liste des ressources et des assignations de tenants pour le pas de temps étudié. Nous avons validé cette approche avec une expérimentation basée sur des données d'utilisation de clients de Bonitasoft.

Les limites de cette approche concernent l'utilisation sur plusieurs pas de temps. Le nombre de migrations concernant tous les tenants confondus, une utilisation itérative peut provoquer des migrations successives d'un même tenant. Chaque migration provoquant des indisponibilités, et ralentissements au niveau du tenant migré et des tenants colocalisés, celles-ci se doivent d'être contrôlées finement au niveau des tenants. Toutefois, cette approche constitue une base de référence pour les étapes suivantes.

5.2 Algorithme mono-objectif multi pas-de-temps basé sur la segmentation de séries temporelles

Dans cette section, nous décrivons nos évolutions multi pas-de-temps de l'algorithme de la section précédente. Ce dernier permettait de trouver le front de Pareto (point de vue coût et migrations) pour un pas de temps donné. Nous cherchons dans cette section à trouver des solutions peu onéreuses pour un ensemble de pas de temps, sous contrainte d'un nombre de migrations maximal par tenants. En effet, la considération du nombre de migrations comme un objectif peut poser des problèmes avec une utilisation itérative du premier algorithme. Dans la première sous-section, nous proposons une approche basée sur la segmentation de séries temporelles couplées à une utilisation d'une version modifiée de l'algorithme décrit dans la section précédente, ainsi qu'une modélisation. La deuxième sous-section présente une approche basée sur les algorithmes génétiques améliorant les résultats obtenus par cette segmentation, ou par l'utilisation de solveur d'optimisation linéaire. Nous présentons également une approche sous forme de modèle, alternative à notre heuristique itérative.

5.2.1 Limites de l'utilisation de l'algorithme mono-objectif sur plusieurs pas de temps

Ce travail est une évolution de notre précédent algorithme où nous avons proposé un modèle d'optimisation bi-objectif pour le coût et le nombre de migrations, d'un pas de temps au suivant, ainsi qu'une heuristique efficace. Nous souhaitions généraliser l'algorithme à un nombre de pas de temps arbitraire. Autant l'utilisation des métriques coût et nombre de migrations a du sens pour une optimisation sur un pas de temps, autant pour plusieurs pas de temps ceci est plus compliqué. Une utilisation itérative de notre modèle initial nécessite que nous fassions un choix parmi les solutions du front de Pareto obtenu. Cependant, rien n'assure qu'une solution locale choisie parmi le front de Pareto soit optimale pour l'ensemble des pas des temps.

FIGURE 5.4 – Illustration d'un tenant migré à chaque pas de temps

Un autre problème concerne le critère "nombre de migrations". Celui-ci considère l'ensemble des tenants, or rien n'empêche d'avoir plusieurs migrations du même tenant. Comme nous avons pu voir dans la section 4.3 du chapitre 4, les migrations ont des effets négatifs sur les performances. Avec une utilisation itérative de cet algorithme, nous ne pouvons assurer un nombre de migrations minimal pour chaque tenant. La figure 5.4 illustre ce problème. Dans cet exemple nous avons quatre pas de temps, et trois tenants t1, t2, et t3. Seul t3 voit sa charge évoluer. Elle devient plus grande au pas de temps 2, plus petite au pas de temps 3, et de nouveau plus grande au pas de temps 4. La ressource R1 n'est pas suffisante pour héberger les trois tenants aux pas de temps 2 et 4. Dans ce cas, la solution choisie est d'assurer une migration à chaque pas de temps. Notre algorithme peut se retrouver à toujours migrer le tenant *t3*, ramenant sur lui l'ensemble des défauts de qualité de service.

Afin de répondre à cette problématique, nous nous proposons dans l'algorithme suivant de limiter le nombre de migrations par tenant, sur la durée étudiée. Ceci transforme le deuxième objectif (voir formule 5.2) en contrainte, comme nous le verrons dans la sous-section suivante, et ainsi ramène le problème à un problème mono-objectif, plus simple à résoudre. Toutefois la dimension temporelle est ajoutée et augmente la taille du modèle.

5.2.2 Modèle d'optimisation linéaire avec contrainte quadratique

Le modèle proposé ici est basé sur celui de la section précédente. Nous avons ajouté la dimension temporelle, transformé l'objectif en contrainte, exprimé l'indicatrice décrite dans la fonction objectif de la formule 5.2 en tant que contrainte quadratique.

Soient les variables suivantes :

- $\mathcal{T}$, l'ensemble des types de configurations du cloud, avec t sa cardinalité.
- $\mathcal{I}$, l'ensemble des tenants avec n sa cardinalité.
- $\mathcal{J}$, correspond à $\mathcal{T} \times \mathcal{I}$ l'ensemble de tous les types de configurations associables à chaque tenant. Sa cardinalité est $m = t \times n$.
- C_j, et W_j, respectivement le coût et la capacité de la configuration j, avec $j \in \mathcal{J}$.
- $\mathcal{K}$ représente l'ensemble des pas de temps, de 0 à D, où $D + 1$ est le nombre de pas de temps.
- w_i(k), les besoins du tenant i durant le pas de temps k.
- ${x_j}^i(k)$, l'assignation du tenant i à la configuration j durant le pas de temps k.
- $y_j(k)$, l'activation ou non de la configuration j durant le pas de temps k.
- M est le nombre maximal de migrations défini pour l'ensemble des tenants durant les pas de temps étudiés.

$$\min \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K}} C_j y_j(k) \tag{5.7}$$

Nous avons les contraintes suivantes :

$$\forall i \in \mathcal{I}, \forall k \in \mathcal{K} \sum_{j}^{j \in \mathcal{J}} {x_j}^i(k) = 1 \tag{5.8}$$

$$\forall j \in \mathcal{J}, \forall k \in \mathcal{K} \sum_{i}^{i \in \mathcal{I}} w_i(k) {x_j}^i(k) \leq W_j y_j(k) \tag{5.9}$$

$$\forall i \in \mathcal{I} \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K} \setminus \{D\}} {x_j}^i(k) {x_j}^i(k+1) \geq |\mathcal{K}| - M \tag{5.10}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}, {x_i}^j(k) \in \{0,1\}, y_j(k) \in \{0,1\} \tag{5.11}$$

5.2.3 Optimisations et modifications de l'heuristique pas-de-temps pour l'utilisation itérative

Comme nous avons pu le voir dans la section précédente, notre algorithme « pas de temps »est efficace et très proche de l'optimum. Nous avons donc choisi de le réutiliser pour l'utilisation itérative, moyennant quelques modifications. Nous avons besoin d'un moyen de limiter le nombre de migrations par tenant et de limiter le nombre de migrations à une « bonne »solution au lieu de considérer l'ensemble du front de Pareto. Considérer itérativement notre algorithme nous amène à choisir pour chaque pas de temps quel tenant migrer parmi la liste des tenants autorisés à migrer. Nous avons basé notre approche sur ce constat.

Le principe que nous avons pris est de découper le problème en deux parties : déterminer précisément à quel pas de temps il serait intéressant de faire migrer un tenant, et lancer notre algorithme de manière à ce que les tenants ne migrent qu'au moment attendu. Nous avons modifié les entrées de notre algorithme 1 ainsi que des routines internes, en ajoutant un paramètre supplémentaire : la liste des tenants autorisés à migrer. Le principe ici est de n'autoriser la migration que des tenants autorisés, et de garder les autres tenants sur leur ressource actuelle. Les ressources ayant encore des tenants restent bien entendu "verrouillées". Le principe général est simplement d'ajouter des conditions de vérification au niveau des différentes étapes, tout en gardant le principe des combinaisons de ressources. Les étapes de *repacking*, et de *bin packing* sont plus particulièrement concernées. A des fins d'optimisation, nous avons également ajouté une étape éliminant du résultat de la somme de sous-ensemble les ressources contenant exclusivement des tenants inamovibles.

Le deuxième point que nous avons tenu à traiter concerne les performances de l'algorithme. Notre algorithme atteignant des temps de résolution de près d'une heure pour 40 tenants, il n'était pas envisageable de l'utiliser dans sa version initiale. Or, nous avons expliqué que pour un plus grand nombre de tenants, la somme de sous-ensembles, décrite dans l'algorithme 2, prend la majorité du temps. Ceci est du au fait que nous générons et considérons l'ensemble des possibilités à l'aide d'un algorithme récursif de force brute, en filtrant celles correspondant à notre cas de figure. Afin d'obtenir des meilleures performances, nous avons conçu une nouvelle version de l'algorithme, que nous avons repris de Isa et al. [118]. Il s'agit ici d'une version approchée, leur expérimentation montrant qu'ils peuvent obtenir 77% des sous-ensembles réels. Il est également possible dans cet algorithme de préciser le nombre maximal d'itérations, ceci permettant d'accélérer les traitements, au détriment du nombre de résultats obtenus. De plus, comme nous le verrons dans la section des expérimentations, ceci nous a permis de considérer un bien plus grand nombre de tenants.

L'ensemble de ces points nous permet d'avoir une version efficace et utilisable de notre heuristique itérative. Toutefois un nouveau paramètre est à déterminer : l'ensemble des heures de migrations des tenants. Nous nous appliquons à présenter notre approche basée sur les « stratégies de migration »dans les paragraphes suivants.

Nous nommons *stratégie de migration* l'ensemble des pas de temps où chaque tenant d'un ensemble de tenants peut migrer. Pour l'ensemble des pas de temps non concernés par celle-ci pour un tenant donné, le tenant reste sur sa ressource actuelle. Soit une *stratégie de migration*

l'ensemble des $h_i(k)$ avec $0 \leq k \leq D-1$ où chaque tenant i est autorisé à migrer. $h_i(k)$ est égal à 0 si le tenant n'est pas autorisé à migrer entre les pas de temps k et $k+1$, et égal à 1, s'il est autorisé. L'équation 5.12 décrit le nombre maximal de migrations.

$$\forall i \in \mathcal{I} \quad \sum_{k}^{k \in \mathcal{K} \setminus \{D\}} h_i(k) = M \tag{5.12}$$

Nous souhaitons trouver les meilleurs valeurs pour chaque $h_i(k)$ respectant le nombre maximal de migrations pour obtenir le meilleur coût.

Une fois que les pas de temps correspondant aux différentes migrations sont déterminés, nous devons choisir le niveau de capacité requise pour chaque tenant. Comme notre algorithme ne considère pas simultanément plusieurs pas de temps, nous avons choisi d'allouer une capacité fixe pour chaque tenant, pour l'ensemble de chaque période entre les migrations.

Nous appelons $P_i(m)$ la capacité durant la période entre les migrations m_1 et m_2. Afin d'éviter des situations où un tenant serait surchargé durant des pas de temps où il ne doit pas migrer, nous avons considéré la capacité maximale requise parmi les pas de temps concernés, ainsi que le montre l'équation 5.13. Un exemple est présenté dans la figure 5.5 . Ces capacités sont utilisées en lieu et place des capacités initiales des tenants dans notre algorithme pas de temps restreint.

$$\forall i \in \mathcal{I}, \forall m_1 \leq k < m_2, P_i(m_2) = \max_{m_1 \leq k < m_2} (w_i(k)) \tag{5.13}$$

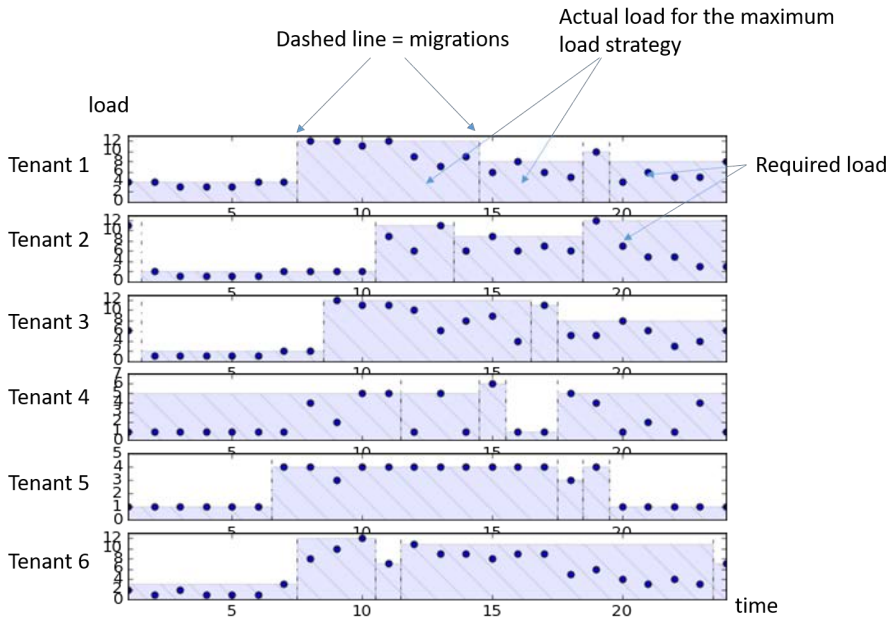

FIGURE 5.5 – Stratégie de migration de 6 tenants sur 24 pas de temps

Comme le test de chaque stratégie de migration nécessiterait trop de temps, nous avons besoin d'un moyen efficace pour déterminer laquelle utiliser. Celle-ci doit donner de meilleurs résultats qu'une approche naïve - que nous présenterons dans la section 5.2.7, et respecter la contrainte sur le nombre de migrations de chaque tenant.

5.2.4 Segmentation de séries temporelles

Notre but est de trouver une stratégie de migration nous permettant d'obtenir une distribution bon marché de ressources et l'assignation des tenants sur celle-ci, pour l'ensemble des pas de temps étudiés.

Nous proposons ici une méthode permettant d'identifier de telles stratégies de migration. Nous avons pris comme postulat que les variations de charge vont produire le besoin de migrations. Celles-ci vont déterminer à quel moment nous devons déclencher la migration d'un tenant. Si par exemple un tenant nécessite un débit de 10 tâches par secondes entre minuit et 6h, et puis un débit de 50 tâches par secondes entre 6 heures et 12h, le meilleur moment pour migrer sera à 6h. Notre approche est pour k migrations de trouver un moyen de fragmenter la série temporelle en $k+1$ fragments, d'une manière à ce que chacun des fragments ait une charge minimale. Les techniques de segmentation de séries temporelles concernent ce genre de problèmes.

La segmentation de séries temporelles est une étape de traitement, et une tâche principale dans plusieurs variétés de « data mining ». Elle est généralement utilisée comme un outil d'analyse de tendance, comme un moyen de réduire la dimension de problèmes, comme un moyen pour discrétiser une série temporelle [119].

Ce dernier point nous intéresse tout particulièrement, comme nous souhaitons trouver un moyen de discrétiser la série temporelle décrivant la charge des tenants. Celle-ci serait découpée en un nombre de fragments, séparés par des migrations.

Les algorithmes les plus connus, basés sur le concept d'approximation linéaire par morceaux (Piecewise Linear Representation) sont à l'origine reportés par Keogh et al. [120] : les approches top-down, bottom-up, et sliding-window (fenêtre glissante). Les deux premières sont des approches offline, similaires aux techniques de regroupement hiérarchique mais appliquées à une série temporelle. On peut comparer l'approche top-down à une classification descendante hiérarchique, et l'approche bottom-up à une classification ascendante hiérarchique. La troisième technique basée sur les fenêtres glissantes considère les pas de temps de manière itérative depuis l'observation la plus vieille jusqu'à la dernière obtenue, et en agglomérant les observations en segments. La classification d'observations dans un nouveau segment est décidée si la distance entre les observations et la valeur du segment dépassent un seuil, qui doit être précisé. Nous nous concentrons sur les deux premiers algorithmes, en utilisant comme critère d'arrêt le nombre de segments.

Le principe général des approches top-down et bottom-up est de récursivement séparer (top-down) ou fusionner (bottom-up) des ensembles d'observations consécutives dans la série temporelles, de manière à ce que les segments gardent une valeur proche de celle des observations d'origine, et ce jusqu'à un critère d'arrêt. Comme on peut le voir dans [120] et [119], la version principale de ces algorithmes utilise l'approximation linéaire par morceaux. Celle-ci calcule les valeurs de chaque segment à l'aide d'une fonction affine trouvée par régression linéaire sur l'ensemble des observations de chaque segment. Sont retenus les segments dont la distance entre la fonction affine et les observations est la plus faible. On peut également trouver des approches basées sur la considération de la moyenne (de manière constante) de l'ensemble des observations d'un segment en lieu et place de la fonction affine trouvée par régression.

Ces approches sont intéressantes pour représenter finement une série temporelle. Toutefois, notre cas est un peu différent : comme nous l'avons précisé dans la partie précédente avec l'équation 5.13, nous souhaitons considérer pour chaque pas de temps hors-migration la besoin maximal du tenant avant la future migration. Nous avons donc proposé une approximation linéaire différente, basée sur la considération de la valeur maximale de chaque segment, de manière constante.

Une fois les segments obtenus par l'étape de segmentation, nous calculons la stratégie de

migration correspondante. Le principe général est d'initialiser l'ensemble des valeurs de la matrice à zéro, hormis pour les pas de temps où il y a un changement de segments où nous initialisons les valeurs à 1. Nous utilisons alors cette stratégie de migration avec l'algorithme pas de temps restreint.

5.2.5 Synthèse de l'algorithme multi pas de temps

Les différentes étapes de notre algorithme sont les suivantes, on peut également les retrouver dans la figure 5.6 :

- Calcul de la stratégie de migration à l'aide d'une segmentation de série temporelle
- Initialisation du pas de temps initial avec la distribution de ressources et de tenants initiale
- Pour chaque pas de temps de $k = 1$ à $k = D$:
  - Lancement de l'algorithme pas de temps restreint, en utilisant la distribution du pas de temps précédent, et la liste des tenants pouvant migrer au pas de temps courant.
  - Sélection de la distribution de ressources et tenants la moins chère, et avec le plus petit nombre de migrations. C'est la distribution que nous allons choisir pour la pas de temps courant.

Il est à noter ici qu'en ne considérant que les tenants à migrer qu'à leurs pas de temps prévus, nous assurons le respect de la contrainte 5.12, ainsi que des autres par le biais de l'utilisation de l'heuristique pas de temps restreinte. Dans la sous-section suivante, nous décrivons les paramètres des expérimentations que nous avons réalisées pour montrer les avantages de notre algorithme.

5.2.6 Paramètres des expérimentations réalisées

Dans le but de tester notre solution, nous avons fait quelques hypothèses, et nous nous sommes appuyés sur des fondations réalistes pour la résolution du modèle. Nous avons besoin d'une bonne estimation des besoins des clients, pas de temps par pas de temps d'un côté, ainsi que d'une estimation des capacités des ressources du cloud d'un autre côté. Pour ce dernier point nous nous sommes appuyés sur les capacités obtenues dans le chapitre 4. Plus précisément nous avons repris les capacités et prix des configurations testées : pour rappel, celles-ci sont basées sur des instances RDS de la famille *r3* (optimisées ressources) pour la base de données, et des instances EC2 de la famille *c4* (optimisées CPU) pour le serveur d'applications. Nous avons effectué nos tests sur la solution BonitaBPM 7.3.2 dans sa version open-source, et utilisé le même processus basé sur des tâches automatiques séquentielles. Les tâches font appel à un script calculant la suite de Fibonacci de manière limitée dans le temps, et mettent à jour certaines variables du processus.

Nous avons alors comparé plusieurs méthodes de segmentations sur les mêmes données de charge des tenants, en terme du coût de la distribution obtenue.

Besoins des clients

Nous avons souhaité tester plusieurs quantités de tenants (plus précisément : 5, 10, 25, 50 et 100) avec plusieurs niveaux de besoins. Afin de pouvoir évaluer les performances de notre algorithme sur des besoins représentatifs de clients, nous nous sommes appuyés sur les mêmes profils minimum et maximum issus de données client anonymisées présentées dans la section précédente. Nous avons ensuite commencé par générer aléatoirement les besoins des tenants pour chaque pas de temps initial, à l'aide d'une distribution uniforme, d'une manière similaire à l'expérimentation de l'algorithme pas de temps.

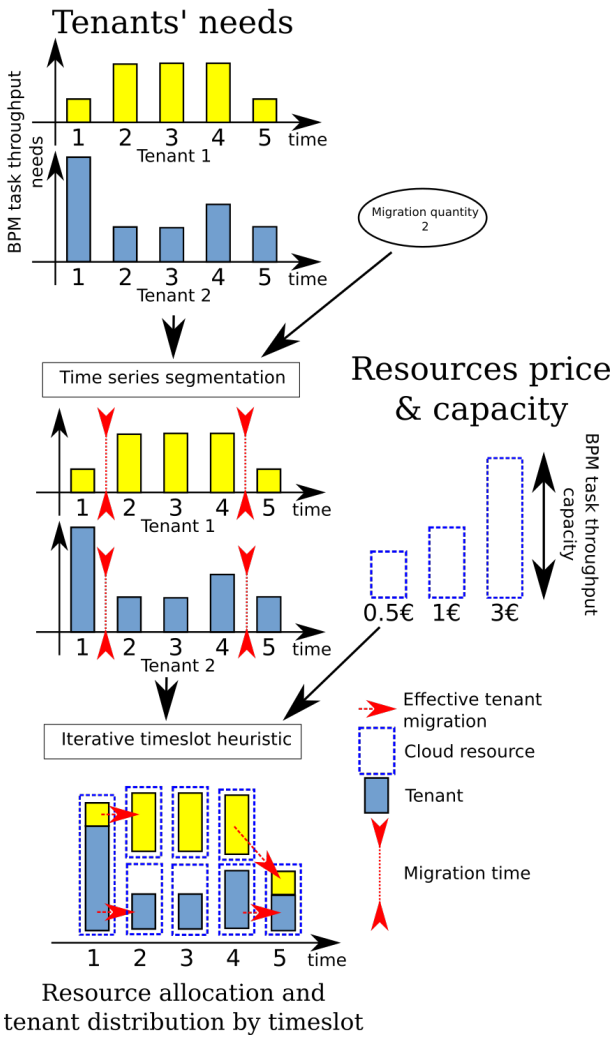

FIGURE 5.6 – Phases de l'algorithme de segmentation

Toutefois, le problème de cette approche est qu'entre les deux bornes définies, les besoins des tenants évoluent de manière totalement chaotique. En général, les besoins des tenants en termes de débit de tâches restent relativement stables, nous avons pu constater cela dans les données des clients. Afin de pallier à ce problème, nous avons ajouté un paramètre que nous avons nommé « *tenant gap* » (intervalle de tenant). Il s'agit d'un pourcentage appliqué sur l'écart entre la borne minimale et maximale des besoins des tenants considérés. Le but ici est de réduire la variation en ne permettant des variations entre pas de temps correspondant à ces bornes restreintes. Le principe que nous avons pris est qu'à chaque pas de temps, nous calculons une valeur aléatoire selon une distribution uniforme, comprise dans l'intervalle correspondant à cette valeur calculée, de manière positive ou négative. Cette valeur aléatoire est ensuite ajoutée au besoin du pas de temps initial, puis à chaque pas de temps itérativement, comme on peut le voir dans l'algorithme 7. Dans le cas où le besoin calculé du client est supérieure à la borne maximale, nous le ramenons à cette valeur. Ceci est réalisé respectivement pour la borne minimale en cas de calcul inférieur à celle-ci.

Par exemple, considérons un tenant E avec un minimum de 5 et un maximum de 120. Pour un *tenant gap* de 0.25, l'écart de besoin entre les heures suivrait une distribution uniforme entre

Algorithm 7 Calcul tenant gap

```
1: procedure TenantGap(duration, min, max, gap)                    ▷
2:     tenantLoad = []
3:     tenantLoad[1] = uniformRandom(min, max)
4:     for timeslot ∈ [2, duration] do
5:         shift = uniformRandom(-(max-min), (max-min))
6:         tenantLoad[timeslot] = tenantLoad[timeslot] + shift
7:         if tenantLoad[timeslot] > max then
8:             tenantLoad[timeslot] = max
9:         if tenantLoad[timeslot] < min then
10:            tenantLoad[timeslot] = min
       return tenantLoad
```

$-0.25(120-5) = -28.75$ et $0.25(120-5) = 28.75$. Les seuils minimal et maximal des besoins du client resteraient ceux du tenant, soit 5 et 120.

On peut garder à l'esprit qu'un *tenant gap* de 1 (soit 100 %) provoquera le même comportement chaotique que celui de l'expérimentation de la section précédente. Un *tenant gap* de 0 provoquera un besoin stable sur l'ensemble des pas de temps, initialisé à la valeur initiale.

Nous avons effectué nos expérimentations avec plusieurs valeurs différentes de *tenant gap*, plus précisément 0.25, 0.5, 0.75 et 1, soit 25 %, 50 %, 75 % et 100 % de variation entre pas de temps. Cinq nombres différents de tenants ont été étudiés, de manière répétée avec différentes distributions pour chaque *tenant gap*. Nous avons utilisé 20 différentes amorces du générateur de nombres aléatoires. Le comportement du générateur de tenants est totalement déterministe : utiliser la même amorce sur le générateur donnera toujours les mêmes ensembles de tenants et de besoins correspondants. Pour chaque paramètre d'entrée de l'expérimentation (l'amorce, les gaps de tenants, et la quantité), nous avons testé plusieurs méthodes de segmentation. Nous les décrivons dans la section suivante.

Méthodes de segmentation et outils

Notre avons voulu déterminer quelle méthode de segmentation et de découpage de tenants donne les meilleurs stratégies de migration. Pour cela nous avons considéré les points suivants :

- utilisation de l'algorithme « top-down »ou « bottom-up »
- considération de la distance « regression linéaire » (Linear Regression), « moyenne constante » (ConstantPieceWise), « maximum constante » (ConstantMaxPiecewise). Cette dernière distance est un de nos apports de ces travaux.
- considération de l'ensemble des tenants comme un tout, ou de manière individuelle. Nous avons tenu à observer la différence entre la considération de l'ensemble des besoins des tenants de manière groupée, et la considération des tenants de manière individuelle. L'approche individuelle est celle décrite dans la section précédente : les stratégies de migrations sont déterminées pour chaque tenant. L'approche groupée prend comme principe l'utilisation d'une même stratégie de migration pour l'ensemble des tenants. La capacité concernée pour la segmentation est représentée par la variable $W(k)$ définie dans l'équation 5.14. Une fois les migrations déterminées à l'aide de cette charge, nous migrons l'ensemble des tenants simultanément à celles-ci.

$$\forall k \in \mathcal{K}, W(k) = \sum_{j}^{j \in \mathcal{J}} W_j(k) \tag{5.14}$$

Nous avons choisi de baser nos pas de temps sur une granularité horaire, et avons expérimenté sur des durées de 2 jours, soit 48 pas de temps. Plusieurs nombres de migrations ont été envisagés : 2, 3 et 4 par jour (soit 4, 6 et 8 pour les 48 pas de temps étudiés). Ce choix s'appuie sur l'étude réalisée dans la section 4.3. Les migrations, selon la méthode employée, provoquent des indisponibilités au niveau du tenant migré, ainsi que des tenants colocalisés.

Pour la partie segmentation, nous avons implémenté une version corrigée et mise à jour de la bibliothèque de segmentation Alchemyst [43] développée par Sandrock[121]. Celle-ci propose des implémentations des algorithmes top-down et bottom-up, avec les distances « régression linéaire »et « moyenne constante ». Nous avons ajouté la distance « maximum constante », comme décrit dans la section 5.2.4 Le code correspondant est libre et accessible [44].

Afin de comparer les performances de notre heuristique, nous avons défini la configuration de référence suivante : pour chaque tenant nous sélectionnons la ressource la moins chère pouvant supporter ce dernier sur l'ensemble des pas de temps. Cette approche - très intuitive et fonctionnelle, est souvent utilisée en production. Nous nommons cette approche, « approche intuitive »(*adapted approach*). Cette méthode permet d'assurer que l'ensemble des besoins des tenants sont assurés.

Nous avons également implémenté le modèle décrit dans la section 5.2.2 à l'aide du solveur Gurobi [45] dans sa version 6.5.1. Toutefois, nous avons pu résoudre le modèle seulement pour les quantités les plus faibles de tenants.

Le tableau 5.4 récapitule l'ensemble des paramètres considérés.

Groupe	Variable	Taille	Valeurs considérées
data	tenant gap %	4	0,25, 0,5, 0,75, 1
data	nombre de graines	20	-
data	nombre de jours	1	2
data	nombre de tenants	6	5, 10, 25, 50, 100, 200
data	nombre de migrations	3	2, 3, 4
segmentation	distance segments	3	moy. const. max. const. régr. linéaire
segmentation	algorithme	2	bottom-up, top-down
stratégie de groupement	besoin des tenants	2	groupé, individuel
algorithme pas de temps	taille somme sous-ensemble	1	1

TABLE 5.4 – Synthèse des paramètres employés dans l'expérimentation

5.2.7 Efficacité de l'approche

Comme nous l'avons expliqué dans la sous-section précédente, nous avons comparé nos résultats avec l'approche intuitive (*adapted approach*). Les résultats correspondant à cette approche sont décrits dans la figure 5.7. Le coût moyen constaté ici est compris entre 370,15 $ pour 5 tenants avec un *tenant gap* de 0,25 à 18853,56 $ pour 200 tenants et un *tenant gap* de 1. Nous pouvons également constater dans cette figure que plus le *tenant gap* est élevé, plus le coût est élevé, pour un nombre de tenants égal. Comme la variation de charge est moins restreinte, la charge maximale a plus de chances d'être élevée, et de nécessiter ainsi des ressources plus onéreuses.

43. `https://github.com/alchemyst/Segmentation`
44. `https://github.com/guillaumerosinosky/Segmentation`
45. `http://www.gurobi.com/`

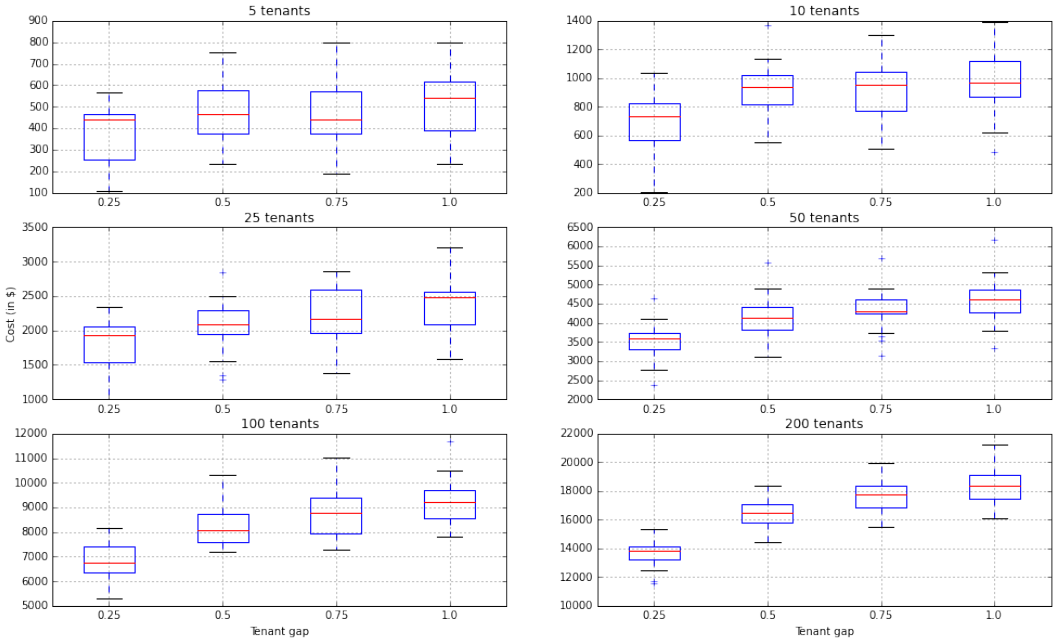

FIGURE 5.7 – Coût de l'approche intuitive en fonction du nombre de tenants et du *tenant gap*

Concernant les résultats de l'algorithme de segmentation, la figure 5.8 présente les résultats moyens de notre heuristique pour l'ensemble des amorces que nous avons testées, ce pour un *tenant gap* de 0,25 et 4 migrations par jour. Nous avons calculé le pourcentage de gain en observant pour chaque expérimentation (celle-ci correspondant à une amorce précise, un *tenant gap*, une quantité de tenant, et un nombre de migrations) le ratio du coût, comparé à l'*approche intuitive*. Nous pouvons remarquer ici clairement que les stratégies donnent des résultats différents. Les approches basées sur l'heuristique top-down (lignes pleines) sont généralement plus efficaces que les approches bottom-up (lignes en pointillés). Les approches groupées (lignes larges) donnent presque à chaque fois de meilleurs résultats comparés aux approches individuelles (lignes fines), excepté pour l'approche top-down/individuel/constant max. Celle-ci donne des résultats proches et parfois supérieurs aux deux meilleures approches groupées. Les meilleurs résultats donnent un gain au-delà de 20 % comparé à l'approche intuitive.

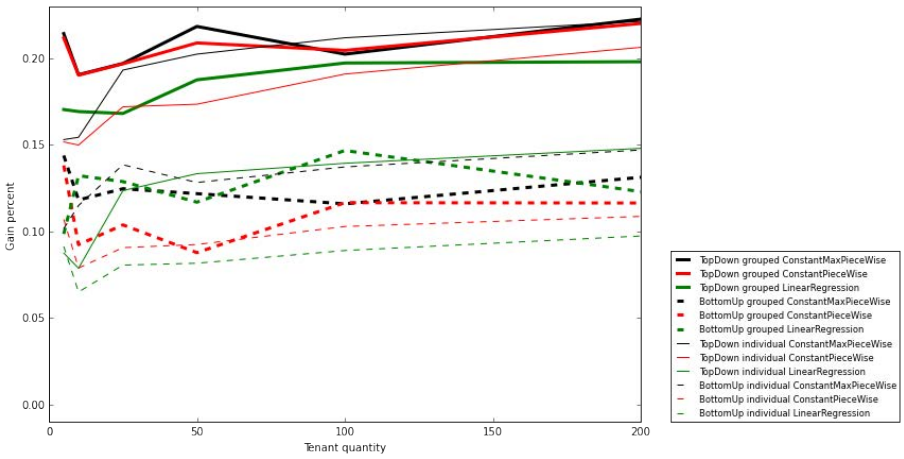

FIGURE 5.8 – Gain en pourcentage de variations l'algorithme comparé à l'approche naïve pour un *tenant gap* de 0,25 et 4 migrations par jour.

La figure 5.9 montre un aperçu global des résultats pour l'ensemble des nombres de migrations et des *tenant gap*. Comme pour le *tenant gap* de 0.25 et les 4 migrations par jour, l'algorithme top-down donne quasiment toujours de meilleurs résultats que l'algorithme bottom-up. Les meilleures approches globale sont en général plus efficaces que les individuelles sur de faibles nombres de tenants ou avec moins de variation (*tenant gap* faible). Les meilleures approches sont les approches top-down/groupé/constante max, top-down/individuel/constante max et top-down/individuel/moyenne max. L'approche top-down/individuel/constante max donne les meilleurs résultats dans beaucoup de bas de figure (pour plus de 2 migrations). Comme on peut s'attendre, nous obtenons de meilleurs résultats quand plus de migrations sont permises (entre 5 et 10 % pour 2 migrations, et entre 15 et 25 % pour 4 migrations). On peut également constater qu'une variation plus haute des besoins des clients donne de moins bons résultats.

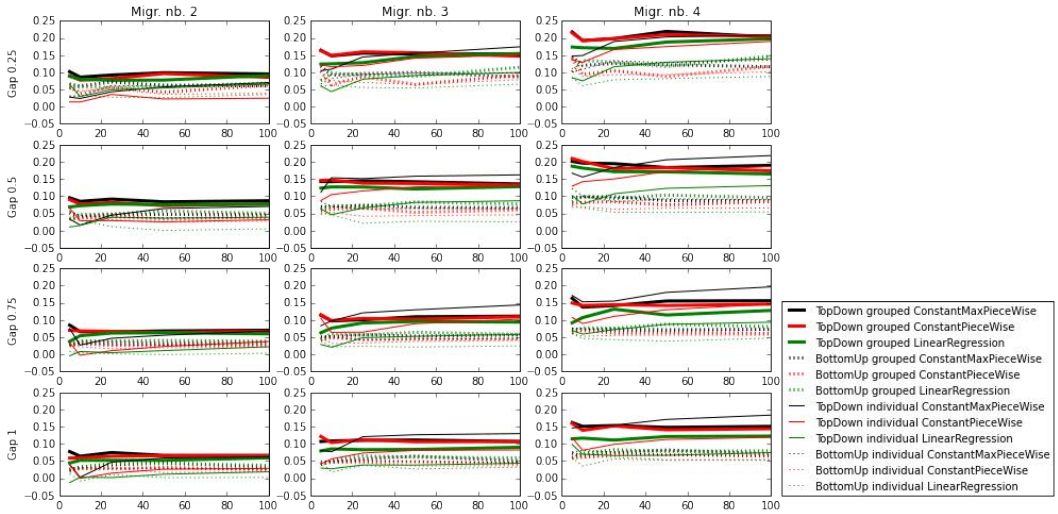

FIGURE 5.9 – Gain en pourcentage de l'algorithme par rapport à l'approche naïve comparé au nombre de tenants

La figure 5.10 montre le temps de calcul de l'heuristique itérative avec différents paramètres. Celle-ci reste relativement stable, pour une stratégie donnée et un nombre de tenants. Les approches individuelles sont toujours plus longues. La durée semble multipliée par 3 ou 4 chaque fois que le nombre de tenant double.

Nous avons voulu comparer nos résultats avec la solution optimale calculée à l'aide du solveur. Calculer les résultats à l'aide de celui-ci est très long, et ne donne pas de bons résultats pour plus de 5 tenants, et une durée maximale de 30 minutes, comme il est possible de le constater dans le tableau 5.5. Pour 10 tenants, même 24 heures de calcul ne donnent pas les résultats optimaux : nous avons obtenu un écart MIP de 7 % au mieux dans ce cas alors que notre heuristique permet d'obtenir un gain de 20 % avec une durée moyenne d'au plus 0.4 secondes de temps de calcul. Pour 100 tenants, la durée totale de notre heuristique est de 100 secondes pour une approche groupée.

On peut remarquer la corrélation entre le nombre de migrations et le gain. Les résultats nous montrent que l'algorithme top-down est bien plus efficace que l'algorithme bottom-up, et que la distance constante maximale donne en général de meilleurs résultats que les autres, suivie par la distance constante moyenne. Les bons résultats des approches globales peuvent être expliqués par l'approche orientée ressource de notre heuristique itérative. Pour rappel, mis à part pour les tenants surchargeants et « unfit », nous considérons le retrait de ressources pour migrer les tenants, ce qui provoque des migrations groupées. La très bonne performance de la distance maximale constante montre l'intérêt dans notre cas d'utilisation de ce type de distance. Toutefois ce dernier fonctionne bien pour un nombre suffisant de tenants.

Nous avons montré que notre heuristique est rapide, même quand le nombre de tenants grandit, et permet des économies substantielles pour les fournisseurs de BPMaaS. Un gain de 20 % sur l'approche intuitive pour 100 tenants correspond à une économie d'en moyenne 1373,48 $ pour deux jours. De plus, une fois les segmentations calculées, il est possible de comparer plusieurs stratégies, vu que le temps de

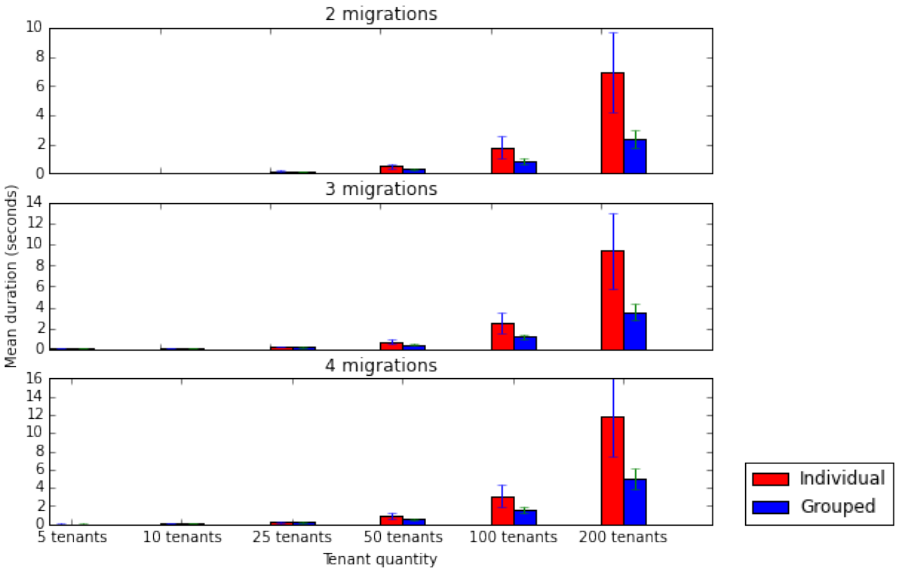

FIGURE 5.10 – Durée moyenne en secondes pour les approches individuelles et groupées, pour chaque nombre de tenants. L'écart-type est représenté sur chaque barre, celle-ci représentant le nombre de secondes.

Qté.tenants	Tenant gap	Moy.	Nb.Migr.	Durée solveur	Gain solveur	Moy. MIP gap
5	0.25	419.99	2	1800	41,05%	3,57%
5	0.25	421.91	3	1800	48,28%	3,45%
5	0.25	421.91	4	1800	51,64%	3,91%
10	0.25	685.82	2	1800	3,18%	48,33%
10	0.25	685.82	3	1800	1,28 %	52,68%
10	0.25	685.82	4	1800	1,28 %	55,33 %

TABLE 5.5 – Résultats du solveur pour les 10 premières amorces. La moyenne de MIP gap est le ratio entre la borne inférieure et la borne supérieure.

calcul de l'heuristique itérative reste relativement peu élevé (pour 200 tenants, une approche individuelle durera en moyenne 11,75 secondes et 5,02 secondes pour une approche groupée.

Les approches individuelles sont plus intéressantes à utiliser dans un environnement de production. En effet, même si nous n'avons pas considéré cette contrainte, migrer l'ensemble des tenants en même temps peut avoir des effets négatifs sur les performances des tenants : l'ensemble des données serait migré simultanément, ceci pouvant occuper une grande portion de la bande passante du réseau. De plus, pour une stratégie de migration donnée, une stratégie individuelle donnera toujours le même résultat : une fois la segmentation effectuée, il n'est pas nécessaire de recalculer celle-ci pour la réutiliser avec d'autres clients. Celle-ci peut être calculée en dehors de la phase d'attribution. Ceci permet de réduire le temps de calcul, en ne considérant que l'heuristique.

5.2.8 Conclusion

Dans cette section, nous avons proposé un nouveau modèle linéaire, ainsi qu'une heuristique simple à implémenter pour l'allocation de ressources et l'assignation de tenants à ces dernières. Nous avons testé nos algorithmes à l'aide de données basées sur les informations d'utilisation de clients, et montré les résultats intéressants fournis par nos algorithmes. Toutefois, comme nous avons pu voir pour un faible nombre de tenants, ces résultats sont améliorables. Nous proposons dans la section suivante une méthode

basée sur une méta-heuristique permettant d'obtenir des résultats moins onéreux.

5.3 Amélioration des résultats à l'aide d'algorithmes génétiques et alternative linéaire à l'heuristique

L'heuristique présentée dans la section précédente permet d'obtenir de bons résultats initiaux, avec des algorithmes faciles à maîtriser. Toutefois comme nous avons pu le voir pour un faible nombre de tenants, il possible d'améliorer ces résultats. Nous proposons dans cette section une méthode permettant d'améliorer les résultats obtenus, ainsi qu'un *modèle restreint* d'optimisation fournissant une alternative pérenne à l'heuristique itérative.

5.3.1 Les stratégies de migrations comme axe d'amélioration

Comme nous l'avons précisé dans les sections précédentes la taille du problème empêche une résolution aisée. Nous avons proposé une première approche s'appuyant sur le concept de stratégies de migrations. Pour rappel, celles-ci représentent pour chaque tenant et chaque pas de temps son « autorisation »à être migré sur une ressource différente. L'utilisation d'une étape de segmentation comme nous l'avons décrite dans la section précédente donne des résultats intéressants, toutefois celle-ci reste une première proposition. Il doit être possible de trouver des stratégies de migrations fournissant de meilleurs résultats.

C'est notre approche dans cette section : nous pensons pouvoir ainsi faire de plus grandes économies tout en respectant les contraintes. Un moyen efficace d'améliorer nos résultats est l'utilisation d'une approche basée sur les métaheuristiques. Le principe des métaheuristiques est d'améliorer itérativement une population à l'aide d'heuristiques, dans le but de converger vers un optimum. Ce type d'approche nous permet de réutiliser nos algorithmes précédents en améliorant les résultats. Nous avons choisi d'utiliser un algorithme génétique pour itérativement améliorer les résultats fournis par la première étape de segmentation, tout en utilisant l'heuristique itérative, ou d'autres méthodes capables de prendre en entrée une distribution de ressources et tenants, une série de capacités et de prix de types de ressources, et un nombre de migrations.

La figure 5.11 présente globalement l'approche envisagée. Une population comprise de stratégies de migrations issues de segmentations est fournie en entrée comme population. Nous y ajoutons également des stratégies de migrations aléatoires. Dans celles-ci nous avons choisi totalement au hasard les pas de temps de migrations, tout en respectant le nombre total fourni en entrée. La fonction de fitness retenue correspond au coût de l'ensemble des ressources actives pour chaque pas de temps. Pour arriver à calculer cette valeur, nous utilisons notre heuristique itérative, décrite dans la section précédente, ou un résolution du modèle restreint avec les valeurs correspondantes, que nous décrirons en section 5.3.4. Les différentes étapes (mutation, croisement, sélection) caractéristiques aux algorithmes génétiques sont alors déclenchées. Les stratégies de migrations considérées comme étant les plus à même de faire poursuivre l'algorithme vers un résultat intéressant sont alors gardées dans la population.

5.3.2 Représentation des stratégies de migration et algorithme génétique

Afin de définir les différents individus, nous avons choisi une représentation correspondant à ce qu'attend un algorithme génétique standard. Celle-ci correspond à une vectorisation des stratégies de migration, comme on peut le voir dans la figure 5.12.

5.3.3 Opérateurs spécifiques à notre problème de l'algorithme génétique

Nous décrivons dans cette section les différentes étapes que nous avons retenues dans notre approche basée sur les algorithmes génétiques. Un schéma précisant les détails de ces étapes est présenté en figure 5.13. Comparé à une implémentation standard d'algorithme génétique, nous avons échangé deux phases : la phase de croisement, et celle de mutation. Ceci est dû à la nature de notre opérateur de mutation, s'appuyant sur les résultats partiels de l'évaluation du score de fitness de l'individu (le coût correspondant à cette stratégie de migration). En effet, la phase de croisement peut générer des stratégies de migrations

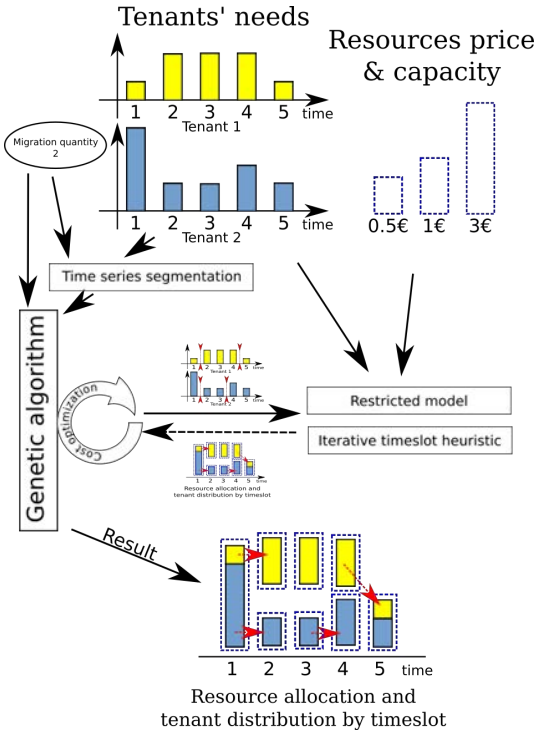

FIGURE 5.11 – Algorithme génétique pour l'optimisation des stratégies de migrations.

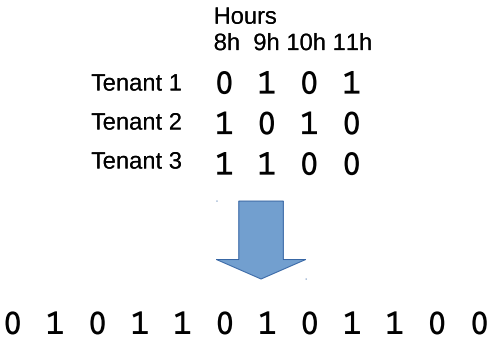

FIGURE 5.12 – Exemple d'un individu.

inconnues (dont le coût n'a pas encore été calculé). Dans notre cas, les « parents »de la population initiale sont mutés en lieu et place de la « descendance »de la population finale.

Initialisation de la population

La génération d'individus aléatoires est souvent utilisée pour initialiser la population. Nous avons tenu à y ajouter des individus plus proches de la solution optimale. Utiliser des individus plus proches de solutions optimales peut permettre de diminuer le nombre d'itérations nécessaires pour atteindre des solutions intéressantes. Pour cela, nous nous sommes appuyés sur les résultats des différents types de segmentations décrits dans la section 5.2.4. Plus précisément, celles-ci correspondent à l'ensemble des combinaisons d'heuristique Top-Down et Bottom-up, de distances constante maximum, constante linéaire, et de régression linéaire, et enfin d'approches individuelles et groupées. Ceci nous a permis d'initialiser la population avec 12 individus différents.

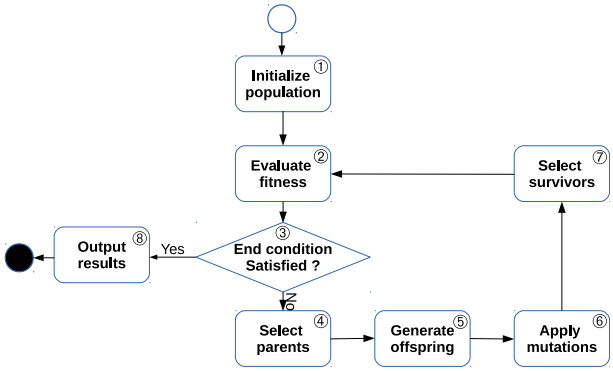

FIGURE 5.13 – Phases de notre algorithme génétique

Algorithm 8 Génération d'un individu aléatoire contraint

```
1: procedure RANDOMINDIVIDUAL(tenantQty, timeslotQty, migrationsNumber)
2:     array = ∅
3:     for i = 1 → tenantQty do
4:         migrationStrategy = [0] * timeslotQty    ▷ tableau de longueur timeslotQty initialisé à zéro
5:         nb ← 1
6:         while nb < migrationNumber do
7:             h ← random(timeslotQty)
8:             if tenantMigrationStrategy[ht] ≠ 1 then
9:                 migrationState[ht] ← 1
10:                nb = nb + 1
11:        migrationStrategy ← array ∪ tenantMigrationStrategy
       return array
```

Nous avons ensuite complété cette population à l'aide d'individus générés aléatoirement mais respectant les contraintes, plus particulièrement celle du nombre de migrations. Nous avons retenu ce principe de respect des contraintes et de nombre de migrations fixe dans l'ensemble des phases. L'algorithme spécifique pour la génération de ces individus aléatoires contraints est décrit dans l'algorithme 8. Les paramètres en sont la quantité de tenants, le nombre de pas de temps étudié, et le nombre de migrations. La fonction renvoie un tableau contenant les stratégies de migration spécifiques de chaque tenant concaténées telles que nous l'avons décrit plus haut. Chaque tenant aura exactement le nombre de migrations prévu.

Évaluation du score de fitness

Nous voulons trouver la stratégie de migration la meilleure marché. Le score de fitness correspond à la somme pour l'ensemble des pas de temps du coût des ressources actives. Pour l'évaluer, il est possible d'utiliser n'importe quel algorithme prenant en paramètre la distribution de ressources et de tenant initiale, la capacité et le prix des types de ressources, et le nombre de migrations permis pour l'ensemble des tenants. Le résultat doit être une distribution de ressources et de tenants pour chacun des pas de temps. Le calcul du coût total se fait en sommant le coût par pas de temps de chaque ressource active. Dans notre cas nous pouvons utiliser notre heuristique itérative décrite en section 5.2.3, ou une résolution à l'aide d'un solveur du modèle décrit en section 5.3.4.

Au vu du nombre d'individus potentiels à évaluer, nous avons implémenté des optimisations pour éviter de recalculer des éléments déjà réalisés. Le résultat obtenu pour une stratégie de migration peut être stocké en mémoire, et réutilisé. Ceci est également valable pour des résultats partiels. Si deux stratégies de migrations commencent par les mêmes valeurs pour l'ensemble des tenants, les distributions des ressources et tenants pour les pas de temps communs seront identiques, il est inutile de les recalculer. Nous avons donc gardé en mémoire l'ensemble des distributions correspondant à chaque stratégie de migration après

lancement du calcul du coût, et assorti une phase préalable initialisant les distributions à celles retenue en mémoire, jusqu'au pas de temps correspondant au point de divergence. Les calculs sont alors lancés à partir de ce dernier.

Condition d'arrêt

Nous avons utilisé une condition d'arrêt temporelle. Après un temps fixé (par exemple 10 minutes), l'algorithme est arrêté et les stratégies de migrations et coûts correspondant aux individus sont retenus comme résultats finaux. L'individu ayant le coût le plus faible est gardé comme résultat de l'algorithme génétique. Cette approche permet de limiter la durée de l'algorithme, celui-ci pouvant tourner pendant une très longue durée, et nous a permis ensuite de comparer les différentes approches.

Sélection des parents

Pour cette étape, nous avons utilisé une sélection par rang. Celle-ci fait partie des sélections les plus utilisées. Pour des raisons de temps de calcul nous n'avons pas pu tester d'autres méthodes de sélection. Le principe général de cette méthode de sélection est de classer l'ensemble des individus par rang, et de sélectionner avec une plus grande probabilité les individus de rang moins élevé, et ce de manière proportionnelle au rang obtenu. L'individu avec le coût le moins élevé aura pour rang 1, celui avec le deuxième coût le plus élevé le rang 2, et ce jusqu'au bout de la liste des individus. Le individus sélectionnés seront ensuite utilisés durant la phase de croisement.

Mutation

Dans l'approche classique, la mutation met à jour les individus de manière aléatoire en modifiant les bits des individus concernés de 0 à 1 ou de 1 à 0 selon une probabilité correspondant au taux de mutation, ceci dans le but d'effectuer de la recherche locale ou de sortir d'un minimum local.

Toutefois dans notre cas, ces valeurs sont contraintes : le nombre total de migrations doit rester constant [46]. Nous avons donc proposé deux approches : une première simple et aléatoire assurant le nombre de migrations, et une deuxième s'appuyant sur l'approche orientée ressource et notre heuristique pas de temps restreinte 5.2.3.

La première mutation que nous avons testée correspond à une application de l'opérateur standard à laquelle nous avons ajouté une restriction sur le nombre de bits pouvant être égal à 1 (ceci correspondant aux migrations). Chaque bit égal à 1 est placé à 0 avec une probabilité correspondant au taux de mutation. Dans le cas où un bit est remplacé par zéro un des autres bits à 0 de l'individu au hasard est remplacé par 1. Ceci assure que le nombre de bits à 1 reste constant, et permet d'ajouter le hasard nécessaire. Cette approche est simple, mais a donné toutefois de très mauvais résultats dans nos expérimentations préliminaires, même avec un grand nombre d'itérations.

Le principe de remplacement de ressources de notre heuristique »pas de temps restreinte « utilisé de manière itérative en est la cause. En effet, pour pouvoir libérer une ressource, l'ensemble des tenants hébergés sur celle-ci doivent être déplacés. Or, avec ce type de mutation il est possible d'avoir une mutation provoquant une migration d'un seul des tenants d'une ressource. Nous en avons conclu que la mutation se devait de tenir compte de l'ensemble des tenants présents sur une ressource, et de déplacer au mieux les tenants présents sur une même ressource à un pas de temps donné. La mise en cache des stratégies de migrations, et de l'ensemble des distributions de ressources et de tenants décrite précédemment a permis un accès facilité à ces informations.

La figure 5.14 présente un exemple comparant ces deux mutations. Les individus y sont représentés de manière tabulaire. Nous considérons ici trois tenants numérotés de 1 à 3, pour 8 pas de temps, ceci donnant lieu à des individus de 24 bits. Les tenants sont placés initialement dans la ressource R1 - coûtant 2$ par pas de temps. Au vu de leurs besoins respectifs, il est envisageable de déplacer à tous les

46. Il était envisageable de laisser ce nombre descendre en dessous du nombre de migrations, mais pour des raisons de simplicité, nous avons préféré le garder égal au nombre de migrations.

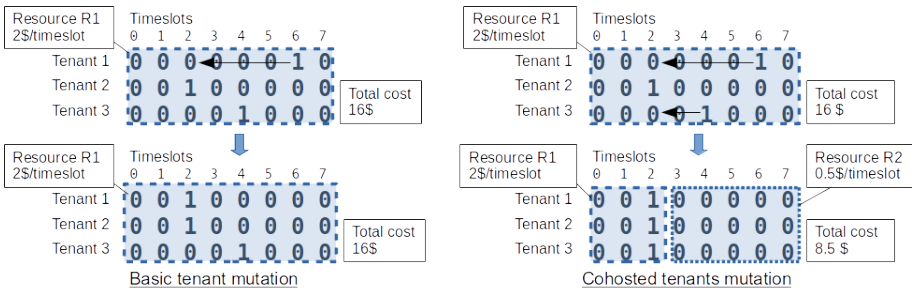

FIGURE 5.14 – Mutation de tenant basique et mutation de tenants colocalisés

pas de temps l'ensemble de ces tenants sur la ressource R2 [47], qui est meilleur marché que R1. Déplacer l'autorisation de migration du tenant 1 du pas de temps 6 au pas de temps 2 ne suffit pas pour libérer l'onéreuse ressource R1. En effet, déplacer les tenants 1 et 2 au pas de temps 2 à la ressource R2 serait plus coûteux au pas de temps 2 que laisser l'ensemble des tenants sur la ressource R1. Les ressources R2 et R1 seraient actives, ceci coûtant 2.5$ au lieu des 2$ de la ressource R1.

Notre mutation de tenants colocalisés (hébergés sur la même ressource) a sa phase de sélection de migration similaire à celle de la mutation basique de tenants. Pour une autorisation de migration considérée (bit à 1), elle consiste à déplacer les bits correspondant aux tenants hébergés sur la même ressource à un pas de temps au hasard sans autorisation de migration, le même pour l'ensemble des tenants. Toutefois, les migrations ayant provoqué l'assignation des tenants à la même ressource ne sont pas forcément arrivées au même moment pour l'ensemble des tenants hébergés. Il faut modifier les autorisations de migrations ayant provoqué l'assignation de chaque tenant hébergé. Pour cela, il convient alors d'itérer en arrière dans les pas de temps jusqu'à atteindre l'autorisation de migration ayant provoqué l'assignation de ce tenant à cette ressource [48]. Dans le cas où aucune migration passée n'a provoqué l'assignation du tenant à cette ressource [49], nous ignorons ce tenant. Une fois l'ensemble des bits de migrations vers la ressource concernée identifiés, nous les plaçons à zéro, et tentons de placer à 1 le pas de temps ciblé pour tous les tenants colocalisés. Si le bit concerné est déjà à 1, nous ignorons ce tenant [50]. Les cas où nous ignorons les tenants signifient que ces derniers gardent leur bit de migration d'origine à 1. Le but ici est de toujours garder le nombre total de migrations stable.

La partie droite de la figure 5.14 illustre cet algorithme. Nous souhaitons déplacer l'autorisation de migration du tenant 1 au pas de temps 2. Pour cela, nous considérons les tenants 2 et 3, hébergés sur la même ressource. En parcourant les pas de temps dans le passé, on retrouve une autorisation de migration au pas de temps 4. Le bit correspondant est placé à zéro, et le bit du pas de temps 2 à 0. L'autorisation de migration du tenant 2 est déjà au pas de temps 2, rien n'est à faire. Ici, nous obtenons un coût total de 8.5$ au lieu de 16$, et arrivons à faire des économies.

Croisement

La phase de croisement consiste à mixer aléatoirement des individus sélectionnés (parents) et à ainsi produire de nouveaux individus (enfants) possédant des caractéristiques (qu'on espère positives) des deux parents. Un ou plusieurs sites de croisements sont utilisés pour découper une partie des parents et les mêler dans les enfants. Une probabilité de croisement est également utilisée. Les parents correspondants ont été sélectionnés préalablement dans la phase de sélection.

Dans notre cas, nous ne pouvons utiliser l'approche standard de croisement. En effet, nous souhaitons garder le même nombre de migrations par tenants, or nos individus consistent en l'ensemble de toutes

47. Nous partons du principe que R2 a une capacité supérieure à la somme des besoins des tenants, ce quel que soit le pas de temps.

48. Dans ce travail, pour des raisons de simplification, nous avons recherché le premier bit non nul, toutefois une autorisation de migration ne signifie pas forcément que le tenant a été assigné au pas de temps en question.

49. Ce qui signifierait que le tenant est resté sur la ressource d'origine.

50. L'autorisation de migration est déjà bien placée.

les stratégies de migrations de tenants de ces derniers. Un découpage mal avisé pourrait provoquer des dépassements du nombre de migrations.

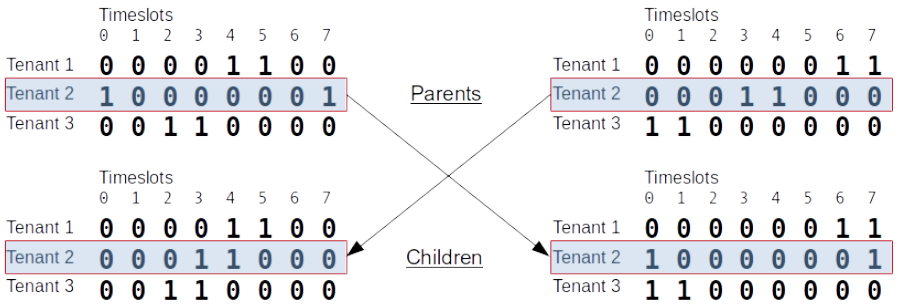

FIGURE 5.15 – Opérateur de croisement

L'approche que nous avons choisie ici est d'échanger les bits correspondant à des stratégies de migrations de tenants pris au hasard entre les deux parents. Le nombre de sites de croisements correspond au nombre de tenants concernés. La figure 5.15 illustre ce principe. Comme dans la figure 5.14, les individus sont représentés sous forme tabulaire, avec 3 tenant et huit pas de temps. Les stratégies de migrations des tenants 2 sont échangées dans les enfants résultants. Ceux-ci seront considérés lors de l'étape suivante.

Remplacement générationnel

Nous avons choisi ici une approche classique. L'ensemble de la population est remplacé par la descendance, mis à part certaines élites parmi la population dont le nombre peut être déterminé. La sélection des élites est effectuée sur le score de fitness des individus, ceux ayant la meilleure valeur (le coût le plus faible) seront gardés dans la génération future. Un taux de croisement est défini, déclenchant un croisement effectif, ou renvoyant les parents d'origine sans modifications en tant qu'enfants.

5.3.4 Alternative linéaire à l'heuristique « pas de temps itérative »

Comme nous l'avons précisé en préambule, un des avantages de cette approche est que n'importe algorithme prenant en paramètre des stratégies de migrations peut être employé. Nous proposons dans cette section une alternative à l'heuristique « pas de temps itérative ».

L'utilisation de stratégies de migrations permet de proposer une version simplifiée de notre modèle linéaire décrit en section 5.2.2. Nous reprenons ici la plupart des variables décrites dans celui-ci. La différence principale est qu'à la place de la contrainte 5.6 concernant les migrations, nous avons besoin seulement de forcer l'égalité entre les valeurs d'assignation entre les différents pas de temps.

Soient les variables suivantes :

— $\mathcal{T}$, l'ensemble des types de ressources, avec t isa cardinalité.
— $\mathcal{I}$, l'ensemble des tenants avec n sa cardinalité.
— $\mathcal{J}$, correspond à $\mathcal{T} \times \mathcal{I}$: l'ensemble des ressources du cloud possibles associées à chaque tenant. Sa cardinalité est $m = t \times n$.
— C_j, et W_j représentent respectivement j, avec j dans $\mathcal{J}$.
— w_i(k), la capacité requise en termes de débit de pas de temps pour le tenant i pendant le pas de temps k.
— $\mathcal{K}$ définit l'ensemble des pas de temps, de 0 à D, où $D+1$ est le nombre de pas de temps.
— ${x_j}^i(k)$, l'assignation du tenant i à la configuration j pendant le pas de temps k.
— $y_j(k)$, l'activation de la ressource j durant le pas de temps k.
— M, le nombre maximal de migrations de tenants entre les ressources du cloud pour tous les pas de temps.
— $h_i(k)$ avec $0 \leq k \leq D-1$. $h_i(k)$ est égal à 0 si le tenant i ne doit pas être migré entre les pas de temps k et $k+1$, est égal à 1, s'il peut l'être. L'ensemble de tous les $h_i(k)$ (pour chaque tenant et

chaque pas de temps) est une stratégie de migration. L'ensemble de tous les $h_i(k)$ pour un tenant i donné est une stratégie de migration de tenant.

— les stratégies de migrations assurent de respecter le nombre maximal de migration par tenants : $\forall i \in \mathcal{I} \sum_{k}^{k \in \mathcal{K}} h_i(k) = M$ où M est le nombre de migrations.

L'objectif de notre modèle est de minimiser le coût de l'ensemble des ressources actives, pour chaque pas de temps.

$$\min \sum_{j}^{j \in \mathcal{J}} \sum_{k}^{k \in \mathcal{K}} C_j y_j(k) \tag{5.15}$$

L'ensemble de ces contraintes doit être respecté :

$$\forall i \in \mathcal{I}, \forall k \in \mathcal{K} \sum_{j}^{j \in \mathcal{J}} {x_j}^i(k) = 1 \tag{5.16}$$

$$\forall j \in \mathcal{J}, \forall k \in \mathcal{K} \sum_{i}^{i \in \mathcal{I}} w_i(k) {x_j}^i(k) \leq W_j y_j(k) \tag{5.17}$$

$$\forall j \in \mathcal{J}, \forall i \in \mathcal{I}, \forall k \in \mathcal{K} | h_i(k) = 0, {x_j}^i(k) = {x_j}^i(k+1) \tag{5.18}$$

$$\forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall k \in \mathcal{K}, {x_i}^j(k) \in \{0,1\}, y_j(k) \in \{0,1\} \tag{5.19}$$

Les équations 5.16, 5.17 et 5.19 sont les mêmes que celles de la section 5.2.2. Elles concernent l'obligation d'un tenant d'être assigné à une et une seule ressource pour chaque pas de temps, et la capacité des ressources de ne jamais être surchargée par ses tenants pour chaque pas de temps concerné.

La différence ici est surtout au niveau de l'équation 5.18. Celle-ci représente le respect des stratégies de migrations. Elles sont définies comme des constantes, correspondant à la variable h. La contrainte d'égalité signifie que pour un tenant i et un pas de temps k, les valeurs d'assignations doivent être identiques entre un pas de temps k et le suivant $(k+1)$. Ces contraintes ne sont définies que pour les pas de temps et les tenants où l'autorisation de migrer de la stratégie de migration est égale à zéro. Dans le cas où celle-ci est à 1, la contrainte n'existe pas. Ceci permet de verrouiller l'ensemble de l'espace en ne laissant comme espace de liberté que les pas de temps où les tenants concernés peuvent être migrés. La généralisation de cette contrainte à l'ensemble des pas de temps et des tenants provoque l'effet attendu, et assure également le respect du nombre de migrations (les tenants étant forcés sur la même ressource dans les autres cas).

Par rapport au modèle décrit dans la section 5.2.2, la résolution de ce modèle est bien plus simple, comme nous le verrons dans la section suivante concernant l'expérimentation.

5.3.5 Expérimentations réalisées

Les expérimentations que nous avons effectuées sont basées sur les mêmes prix et tailles de type de ressources du cloud que dans la section précédente, ainsi que sur le même ensemble de tenants.

Concernant les types de ressource, nous avons considéré un ensemble de 12 ressources, chacun composé de deux instances : des instances EC2 de calcul de la famille *c4* et *m3.medium* pour la couche serveur d'applications, et des instances RDS de bases de données de la famille *r3* et *m3.medium* pour la couche bases de données. Les prix et capacités vont de 0,177$ par heure pour un débit de tâches BPM de 16,4 tâches par secondes, à 4,126$ par heure pour un débit de tâches BPM de 129,279 tâches par secondes. Plus de détails sont indiqués dans la section 4.2.3 du chapitre 4.

Nous avons effectué nos tests pour 10, 25, 50 et 100 tenants, en reprenant la méthode de calcul de débit, les tenants, et les amorces utilisées en section 5.2.6 [51]. Nous avons également utilisé la méthode basée sur les *tenant gap* décrite dans l'algorithme 7. Toutefois, pour des raisons de temps de calcul, nous avons seulement considéré un *tenant gap* de 0.25.

51. Les données sont accessibles à l'adresse suivante : `https://doi.org/10.5281/zenodo.1173617`

Logiciels et matériels employés

Pour l'implémentation de l'algorithme génétique, nous avons employé la bibliothèque Inspyred [122]. Celle-ci est développée en Python, et a pu ainsi s'interfacer aisément avec nos algorithmes existants. Son fonctionnement est extrêmement modulaire, et permet de personnaliser la plupart des opérateurs en permettant de fournir des pointeurs de fonctions. Les opérateurs les plus connus sont généralement présents (par exemple : sélection par rang, par croisement proportionnel au score de fitness, uniforme). D'autres métaheuristiques sont également mises à disposition, telle qu'une implémentation de recuit simulé par exemple. Le développement d'opérateurs spécifiques doit seulement suivre la signature des méthodes présentes. Il nous a été aisé de développer nos opérateurs spécifiques et de les tester.

Nous avons implémenté notre modèle restreint à l'aide de la bibliothèque PulP [123] couplé au solveur Gurobi, comme dans les étapes précédentes.

Paramètres de l'expérimentation

Afin d'obtenir des résultats significatifs et réalistes, nous avons utilisé les paramètres suivants :

- 4 valeurs différentes pour le nombre de tenants : 10, 25, 50 and 100
- durée de pas de temps définie à une heure pour correspondre au modèle de coût d'Amazon Web Services [52]
- nous limitons le nombre de migrations à 4 par jour
- nous considérons une période de 2 jours, soit 48 pas de temps (limitant ainsi le nombre de migration à 8)

De nombreux éléments de l'algorithme génétique peuvent être paramétrés à plusieurs valeurs. Après quelques tests non intensifs, nous nous sommes limités aux paramètres suivants.

- Taille de la population : 20. Nous avons commencé par effectuer des tests avec 10 individus, mais ceci ne couvrait pas l'ensemble des segmentations, et restait bloqué assez rapidement dans un minimum local. Avec 20 individus, compris de 12 issus de la phase de segmentation [53], et 8 aléatoires, les résultats étaient bien meilleurs.
- Nombre d'élites : 5. Nous avons testé plusieurs nombre d'élites, entre 0, 1, 2, et 5. Ce dernier nombre donnait les meilleurs résultats.
- Taux de mutation. Après des tests échelonnés entre 0 et 1, le taux de mutation de 0,4 s'est avéré le plus efficace.
- Nombre de points de mutations (tenants concernés par les mutations). Après plusieurs tests, nous avons choisi de considérer 20% de la population. Ceci signifie que ce nombre sera égal à 2 pour une population de 10 tenants, 5 pour 25 tenants, 10 pour 50 tenants et 20 pour 100 tenants.
- Nous avons limité le temps de calcul de l'algorithme génétique à 600 secondes ou 1800 secondes.
- Dans les cas où nous avons testé une résolution du modèle restreint, nous avons limité la durée de calcul du solveur à 5 secondes. Pour les problèmes de grande taille, ce dernier pouvait se retrouver donner des résultats exacts en plus de temps.

5.3.6 Résultats initiaux obtenus suite aux expérimentations

Dans la figure 5.16, nous montrons le gain de notre approche comparé au meilleur résultat de la segmentation [54]. Le calcul des distributions est effectué à l'aide de l'heuristique restreinte itérative. L'approche intuitive reste toujours identique à celle présentée dans la section 5.2.6.

On peut remarquer deux particularités sur cette figure :

- le nombre d'itérations effectuées dans le temps imparti est inversement proportionnel au nombre de tenants
- pour un même nombre d'itérations, les performances sont meilleures pour un plus faible nombre de tenants

52. au moment de l'expérimentation

53. Les 12 individus correspondent à la combinaison des heuristiques Top-Down et Bottom-Up, des distances constantes linéaires, constantes maximum, et régression linéaire, et des approches groupées et individuelles.

54. La segmentation fournissant les meilleurs résultats peut être différente selon le nombre de tenants.

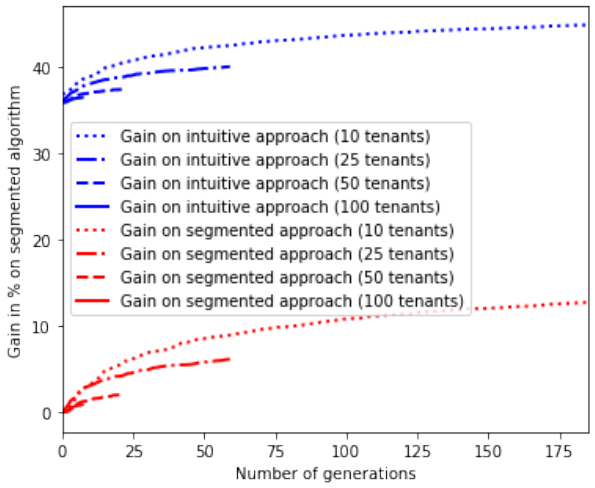

FIGURE 5.16 – Gain moyen de l'algorithme génétique comparé au meilleur individu issu de la phase de segmentation et à l'approche intuitive, pour un temps de calcul de 600 secondes.

Le premier point peut s'expliquer par le fait que l'heuristique pas de temps itérative nécessitera plus de temps pour effectuer les calculs pour un plus grand nombre de tenants. En conséquence, dans ce cas, les évaluations du score de fitness sont plus longues à effectuer, et moins de générations peuvent être calculées. En l'occurrence, pour 10 tenants, plus de 175 générations ont le temps d'être calculées dans les 600 secondes imparties, à comparer aux quelques-unes pour 100 tenants. Le deuxième point que nous pouvons remarquer est que pour un même nombre de migrations, les gains sont moins intéressants pour un grand nombre de tenants. Ceci est probablement dû au fait que la taille de l'espace des solutions est bien plus réduite pour 10 tenants, ce qui permet d'obtenir des résultats intéressants plus rapidement. Ces deux caractéristiques provoquent un gain de plus de 10% de l'algorithme génétique comparé à la meilleure segmentation pour 10 tenants (ce qui amène à 45 % comparé à l'approche intuitive), à comparer à un gain d'environ 1% pour 100 tenants.

Il apparaît nettement ici que l'utilisation conjointe de l'heuristique pas de temps itérative et de l'algorithme génétique est bien plus efficace pour un faible nombre de tenants. C'est pourquoi nous avons choisi d'effectuer nos expérimentations sur des sous-ensembles de la totalité des tenants étudiés où nous avons déclenché l'algorithme, pour ensuite agglomérer les distributions résultantes.

5.3.7 Découpage des tenants en petits groupes et résultats obtenus

Nous avons pris pour principe de séparer les tenants en des groupes sélectionnés aléatoirement. Toutefois nous souhaitions trouver la taille de sous-ensembles fournissant les meilleurs résultats. Pour cela, nous avons choisi de tester plusieurs tailles de groupes pour les différents nombres de tenants, et de comparer la somme des coûts correspondant au meilleur individu de chaque résultat du lancement de l'algorithme génétique sur ces derniers. La figure 5.17 montre les résultats obtenus en utilisant l'heuristique pas de temps itérative comme évaluateur du score de fitness. L'abscisse correspond à la taille de sous-ensembles considérée, et l'ordonnée au gain comparé à l'exécution de l'algorithme génétique sans découpage en sous-ensembles.

On remarquera ici une certaine variation dans les gains, avec clairement de meilleurs résultats pour des groupes de 5 tenants : pour 25, 50 et 100 tenants ceux-ci dépassent 10%. Il faut garder à l'esprit que ces 10% de gain correspondent à 10% du coût total, ce qui n'est pas négligeable.

Concernant l'utilisation de la résolution du modèle restreint, pour des raisons de temps et de coût des ressources, nous n'avons pas pu être capables de tester l'ensemble de tous les paramètres possibles. Par exemple, avec notre implémentation courante, nous avons réussi à obtenir des résultats que jusqu'à 25 tenants. En effet pour plus de tenants, la durée de l'initialisation du solveur et la taille de la mémoire

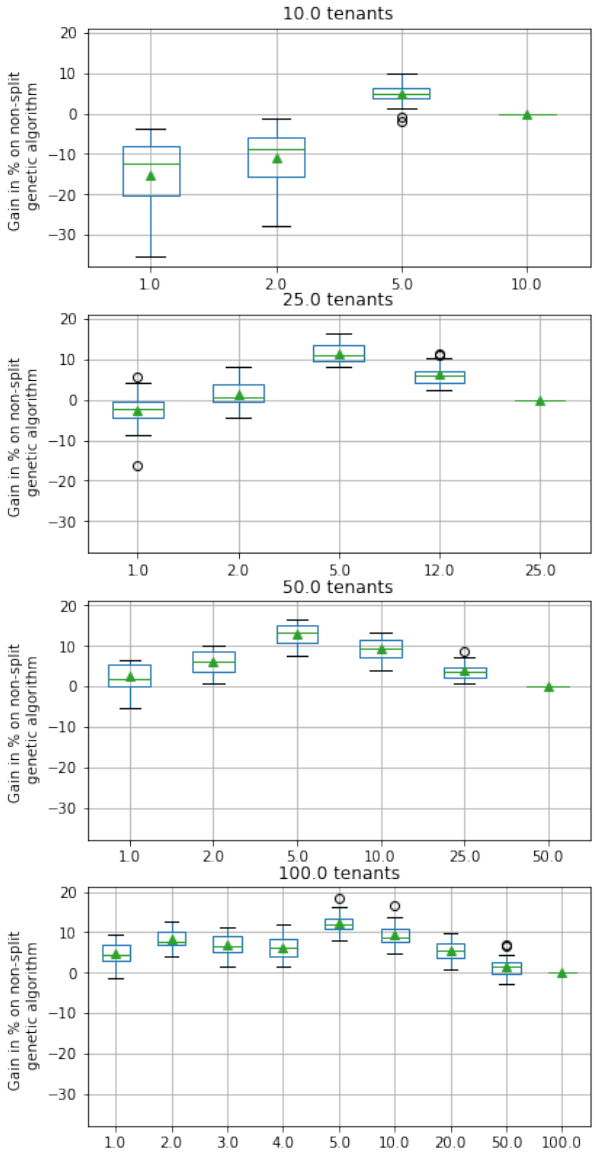

FIGURE 5.17 – Gain comparé à la taille des sous-ensembles de tenants.

nécessaire pour charger en mémoire le problème a rendu impossible la résolution.

Toutefois, les quelques tests que nous avons réalisés avec la résolution du modèle restreint ont donné les mêmes résultats pour 5 tenants, et montrent que ce comportement ne provient pas de l'heuristique pas de temps itérative, mais plutôt du comportement de l'algorithme génétique et à la taille de l'espace des solutions à découvrir. Il est probablement également lié à la capacité des ressources et au besoin des tenants. Dans les étapes suivantes, nous avons effectué nos expérimentation avec des tailles de sous-ensembles de 5 tenants.

5.3.8 Résultats globaux comparés de l'heuristique pas de temps itérative et de la résolution du modèle restreint

Au vu du nombre de tests à effectuer, nous avons circonscrit nos calculs à des ensembles de 50 et 100 tenants. Nous avons tenu à comparer ici l'approche intuitive, la segmentation sur l'ensemble des

tenants, la segmentation sur des sous-ensembles de 5 tenants [55], ainsi que l'algorithme génétique couplé à l'heuristique pas de temps itérative d'un côté, et à la résolution du modèle restreint de l'autre, ce pour 600 secondes et 1800 secondes. Cela nous donne une idée générale du comportement et du gain de nos algorithmes avec ces variations.

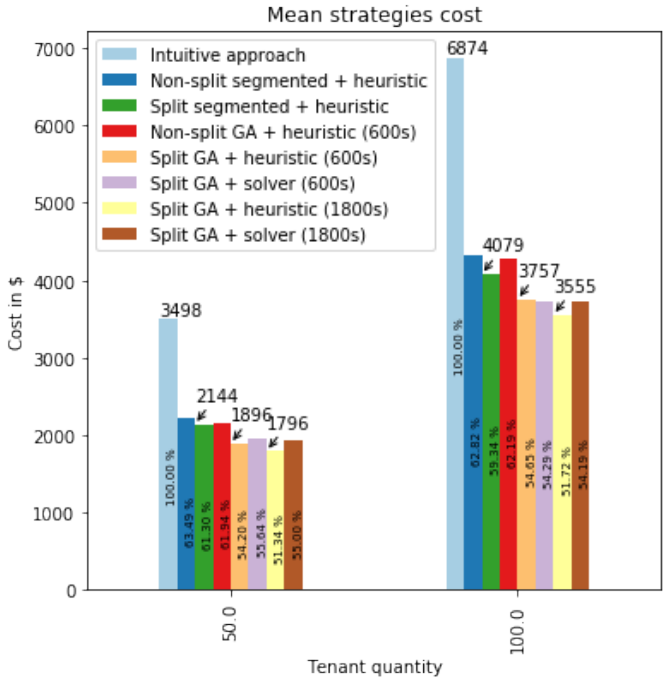

FIGURE 5.18 – Comparaison du coût des résultats obtenus pour 50 et 100 tenants par groupe de 5

La figure 5.18 présente les résultats obtenus, avec le coût total en ordonnée, ainsi que le pourcentage comparé à l'approche intuitive. Pour 1800 secondes de temps d'exécution, l'algorithme génétique couplé à l'heuristique pas de temps itérative donne les meilleurs résultats, avec un coût correspondant à 51,34% de l'approche intuitive pour 50 tenants, et respectivement 51,72% pour 100 tenants. L'utilisation du solveur donne de très bons résultats, quoiqu'un peu plus chers (55% du coût pour 50 tenants, et 54,19% pour 100 tenants). Pour 600 secondes de temps d'exécution les résultats sont plus mitigés entre les deux approches. Ils varient entre 54,2% et 55,64%. On peut remarquer que l'algorithme génétique n'améliore pas beaucoup les résultats pour 1800 secondes comparé à ceux de 600 secondes : cela correspond à 3% pour l'heuristique pas de temps itérative, et 1% pour le solveur. On peut également observer que l'utilisation du découpage en sous-ensemble pour la segmentation sans algorithme génétique permet de gagner un peu. L'heuristique pas de temps itérative fournit de meilleurs résultats pour des plus faibles nombres de tenants, on parle ici d'environ 2% de gain. Avec l'algorithme génétique, le découpage en sous-ensembles donne clairement de bien meilleurs résultats, comme nous avons pu voir précédemment, avec ici un gain de 7.1% pour 50 tenants et 4,69% pour 100 tenants.

Enfin, le gain financier est indéniable. Si on considère l'approche intuitive, nous permettons au fournisseur d'économiser 1702$ pour 50 tenants, et 3319$ pour 100 tenants. Si on compare cela à notre travail précédent (segmentation simple), les économies s'élèvent à 425$ ou 763$, pour -il faut le rappeler- une durée de deux jours selon nos hypothèses.

55. Afin de déterminer à quel point ce découpage en sous-ensemble permet de trouver de meilleurs résultats en dehors de l'algorithme génétique

5.3.9 Conclusion

Notre approche basée sur un algorithme génétique nous a permis d'améliorer substantiellement les résultats de l'algorithme décrit dans la section 5.2. Nous avons également proposé une alternative efficace à notre heuristique « pas de temps itérative ».

Une meilleure adaptation des paramètres de l'algorithme, ou l'implémentation d'autres métaheuristiques telles que le recuit simulé ou la méthode Hill Climbing en lieu et place de l'algorithme génétique pourrait fournir encore de meilleurs résultats.

5.4 Conclusion du chapitre

Nous avons présenté dans ce chapitre nos approches pour l'élasticité des outils BPM dans le Cloud. Celles-ci permettent de grandes économies par rapport à l'approche intuitive souvent utilisée en production, comme nous avons pu voir dans les différentes expérimentations que nous avons présentées. Elles s'appuient sur les hypothèses décrites dans le chapitre 4.

Il faut noter ici que ces algorithmes, basés sur le débit de tâches BPM peuvent être utilisés à l'aide d'autres métriques. Par exemple il serait possible de considérer le débit de processus BPM, ou encore le débit de requêtes HTTP pour d'autres types d'applications Web transactionnelles. Nous pourrions également utiliser d'autres dimensions temporelles pour les pas de temps. Nous avons utilisé des pas de temps d'une heure dans nos tests, mais il serait envisageable d'utiliser des minutes, voire des secondes. Il est également envisageable d'utiliser ces algorithmes « en ligne ». Ceci nécessiterait de les coupler à un module de prédiction du débit de tâches BPM pour les pas de temps suivants. Ceci permettrait à nos algorithmes de s'adapter à des variations non prévues.

Chapitre 6

Bilan et perspectives

Dans cette thèse, notre problématique était d'optimiser les coûts opérationnels d'un fournisseur de BPM en tant que service utilisant une architecture multi-tenant, tout en assurant la qualité de service des clients. Les outils BPM étant des applications Web de nature transactionnelle, les tailles des installations sont limitées en capacité. Nous avons choisi de considérer la distribution des tenants plutôt que la distribution des processus souvent considérée dans l'état de l'art, ainsi que les migrations à chaud, pour des besoins de clients préalablement établis.

Nous présentons dans ce chapitre le bilan de nos contributions, leurs limites ainsi que nos perspectives.

6.1 Bilan des contributions

L'élasticité des outils BPM est un sujet de recherche active. Nous avons pu voir dans le chapitre 3 les nombreuses contributions à ce sujet. Toutefois, aucune ne s'attaque simultanément aux installations multi-tenant et aux migrations à chaud. Nous avons présenté dans ce manuscrit nos contributions, basées sur des hypothèses réalistes. Nous avons présenté ces dernières dans le chapitre 4 : nous ciblons des installations BPM multi-tenant. Les installations BPM ne passant pas à l'échelle facilement, nous avons proposé une approche basée sur la création et la suppression d'installations limitées en capacité. Celle-ci passe par le calcul pour chaque pas de temps de la distribution des ressources à allouer (ou libérer), ainsi que de la distribution de chaque tenant sur chaque ressource. Cette approche sous-entend la migration des installations des tenants. Pour déterminer la taille des ressources et des tenants, nous avons pris comme principe la considération d'une métrique de qualité de service simple, représentative pour l'ensemble des installations, et ayant du sens pour les clients : le débit de nombre de tâches BPM exécutées. Nos approches sont hors-ligne : nous avons besoin pour chaque tenant de leurs besoins dans les termes de notre métrique. Nous avons également besoin d'estimer la taille des ressources en termes de débit de tâches.

Afin de déterminer la taille des ressources en regard du débit de nombre de tâches BPM, nous avons présenté une méthode et un framework permettant d'estimer la taille des ressources en fonction de notre métrique, et avons testé ceci sur le moteur BPM Bonita. Ce framework s'appuie sur des outils libres et est simple à mettre en œuvre. Ceci nous a permis d'obtenir une taille des ressources du cloud, et à été présenté au workshop CloudBPM de la conférence CloudCom 2016 [11].

A cela, nous avons ajouté la considération du nombre de migrations à chaud comme autre métrique de qualité à contrôler. Nous avons mesuré les effets des migrations sur les performances des tenants colocalisés et migrés. Pour cela, nous avons préparé et testé une approche de migrations à chaud, présentée dans le chapitre 4. Afin de tester notre méthode, nous l'avons implémentée sur un moteur BPM multi-tenant d'architecture *shared-schema*. Cette approche a été implémentée et testée sur l'outil Bonita, et est généralisable à d'autres types d'applications. Nous avons ensuite implémenté notre approche d'évaluation des migrations dans un framework basé sur Ansible et Docker Swarm. Cette approche, présentée à la conférence Coopis 2018 [12] a montré que les migrations à chaud ont des effets négatifs visibles sur le temps de réponse des tenants migrés et colocalisés. Ces effets restent raisonnables et limitables comme l'avons vu dans le chapitre 4. C'est pourquoi nous avons pu les considérer comme un critère.

Avec ces hypothèses déclarées, évaluées et justifiées, nous avons proposé des algorithmes et modèles d'optimisation linéaire. Le but de ces derniers est de répondre à notre question d'élasticité : pour chaque pas de temps, quelles ressources allouer et libérer, et quels tenants placer sur chacune de manière à respecter la qualité de service des clients, et à avoir un coût d'infrastructure peu élevé ? La première approche peut être vue comme étant de type réactif, et s'applique à optimiser coût et nombre de migrations simultanément d'un pas de temps au suivant. Celle-ci est basée sur un algorithme de Variable Cost and Size Bin Packing, que nous déclenchons sur une quantité réduite de sous-ensembles des combinaisons de tenants, conclues par des consolidations pour la réduction du coût. Nos expérimentations montrent que notre approche donne de bons résultats, proches du front de Pareto optimal. Nous avons présenté cette approche dans la conférence IEEE Cloud 2016 [13]. La deuxième approche utilise une version étendue de notre algorithme initial pour donner l'allocation des ressources et la distribution des tenants sur ces derniers pour plusieurs pas de temps. Cette version étendue prenant en entrée la liste des tenants pouvant être migrés pour chaque pas de temps, nous utilisons la segmentation de séries temporelles pour fournir la liste de migrations de chaque tenant, en segmentant la charge attendue par ces derniers et considérant que pour chaque changement de segment, une migration doit avoir lieu. Cette approche donne des résultats intéressants : une réduction des coûts d'infrastructure pour les fournisseurs à l'aide d'une méthode simple à mettre en œuvre. L'approche d'optimisation linéaire de son côté reste limitée à une dizaine de tenants, quand nous pouvons traiter des centaines de tenants avec notre heuristique. Cette approche a été présentée dans la conférence BPM 2017 [14]. La troisième approche est basée sur l'amélioration des résultats obtenus par la précédente à l'aide d'un algorithme génétique. Nous nous basons sur l'optimisation du coût correspondant aux stratégies de migration utilisées dans la deuxième approche - correspondant à l'ensemble des autorisations de migrer pour chaque tenant. Pour cela, nous avons proposé des opérateurs spécifiques à notre problème, dont un opérateur de mutation prenant en compte la colocalisation des tenants. Notre algorithme génétique est utilisable avec n'importe quel algorithme hors-ligne prenant en entrée les tenants à migrer, les besoins et capacités et la distribution initiale : nous proposons également un modèle linéaire correspondant à noter heuristique améliorée. Les résultats des expérimentations montrent qu'à l'aide de notre approche il est envisageable d'économiser des milliers de dollars par jour, ce pour des centaines de tenants. Nous avons présenté cette approche dans la conférence ICSOC 2018 [15].

6.2 Limites et perspectives

Avec nos approches d'élasticité, nous avons obtenu des résultats convaincants et encourageants pour les fournisseurs de BPMaaS. Nos méthodes d'allocation de ressources et de distribution de tenants ont toutefois quelques limitations qu'il faut garder à l'esprit :

— dans nos approches, nous considérons seulement l'élasticité horizontale, et la présence d'un seul fournisseur de cloud et centre de données. La considération de fournisseurs multiples de cloud ou de clouds hybrides est plus réaliste, car elle permet aux clients de considérer plusieurs modèles de coût ou leur propre infrastructure. Toutefois des problématiques de latence peuvent apparaître, que l'algorithme doit prendre en considération.

— la considération du nombre de tâches comme métrique fait abstraction du reste de la structure des schémas BPM, tels que les évènements, sous processus, alternatives, etc. Ces derniers peuvent avoir des effets sur les performances des outils BPM [94].

— cette méthode sous-entend l'isolation des performances des tenants dans les outils concernés. Sans celle-ci, un tenant peut se trouver utiliser l'ensemble des capacités du moteur au détriment des autres [124]. Avec les paramètres par défaut du moteur, chaque processus et tâches exécutés dans le moteur ont la même priorité. Le lancement par un tenant d'un nombre élevé de processus diminuerait les performances de l'ensemble de l'ensemble des processus exécutés. De plus, l'utilisation du débit de tâches ne garantit pas un respect du temps de réponse des requêtes des clients, qui serait utilisé pour une approche commerciale. Les temps de réponse sont ce que voient les utilisateurs finaux : des requête trop longues peuvent être gênantes pour les clients. Toutefois, l'implémentation et le paramétrage de limitations du nombre de tâches ou de processus traités au niveau du moteur BPM pourraient permettre de mieux garantir les temps de réponse.

— l'approche hors-ligne utilisée ne considère pas les variations pouvant avoir lieu pendant l'exploitation réelle. Associé au manque d'isolation de performances, ceci pourrait provoquer des résultats non prévus au niveau des temps de réponse pour l'ensemble des tenants.
— les besoins des clients employés pour les expérimentations restent aléatoires, même si elles emploient des seuils maximum et minimum représentatifs. Tester avec des charges de travail issues de données réelles pourraient valider plus précisément nos approches.
— nous considérons le nombre de migrations dans nos algorithmes. Toutefois, comme nous avons pu voir dans le chapitre 4, on peut retrouver différentes durées de migrations selon le nombre de processus ou de tâches actifs. De plus les effets sur les tenants colocalisés sont effectifs et pourraient être pris en compte.

Nos estimations souffrent également de limitations, que nous adressons dans les perspectives :

— nous avons considéré un seul moteur BPM, le moteur Bonita. L'étude d'autres moteurs BPM Open Source tels que Camunda [56], jBPM [57] ou Activiti [58], ou commerciaux tels que IBM [59] ou Bizagi [60] peut être intéressante à considérer pour une estimation généralisée des performances. Nos résultats restent toutefois valides et représentatifs : les moteurs BPM tels que Camunda, jBPM ou Activiti sont basés sur la même architectures que Bonita (Java EE, base de données relationnelle).
— notre framework d'estimation des ressources a été estimé seulement à l'aide d'un processus contenant 20 tâches automatiques séquentielles. Ceci nous a donné un ordre d'idée représentatif des capacités des ressources, mais une estimation plus fine prenant en compte la structure des schémas et les tâches humaines peut être intéressante à considérer.
— notre approche de simulation des utilisateurs utilisée dans notre framework d'estimation des migrations pour la résolution des tâches humaines reste simpliste et ne représente pas le comportement des utilisateurs dans toute sa richesse. Les tâches sont immédiatement récupérées, assignées et clôturées, or dans la réalité les utilisateurs ne géreront pas celles-ci si rapidement. Tester d'autres comportements d'utilisateurs peut être plus réaliste.
— dans son état actuel, notre méthode de migration nécessite de retirer certaines contraintes d'intégrité. Toutefois, ceci n'a pas empêché le bon fonctionnement de l'application.

Nos contributions sont une première étape vers une méthode complète d'élasticité pour solutions SaaS transactionnelles multi-tenant pouvant s'appliquer à n'importe quel type de logiciels de ce type. Nous la décrivons dans la sous-section suivante.

6.2.1 Élasticité pour applications transactionnelles

Nos méthodes sont basées sur le débit de tâches BPM. Toutefois, il est envisageable de les généraliser à d'autres métriques, à condition que celles-ci soient significatives pour l'application. Il est possible de considérer le nombre de requêtes HTTP, de transactions dans la base de données, de messages JMS par exemple. Ceci nécessiterait de mesurer les ressources du cloud en fonction des critères de qualité de service retenus, ainsi que de mesurer le temps de réponse correspondant pour les clients finaux.

Dans les approches d'élasticité, les migrations à chaud ne sont que peu souvent prises en compte dans l'état de l'art, mis à part pour les machines virtuelles et les bases de données en tant que service. Nous montrons dans cette thèse leur utilisabilité dans le contexte d'un BPMaaS. Avec la méthode de migration à chaud que nous avons proposé et les résultats que nous avons obtenus, nous avons démontré qu'il est envisageable pour un fournisseur de SaaS de s'appuyer sur les migrations sans en laisser la responsabilité à la couche base de données, comme le proposent les solutions commerciales telles qu'Azure Elastic Database. Ceci laisse un contrôle plus précis au fournisseur, qui n'a pas a payer pour le service offert par les fournisseurs IaaS et PaaS. La problématique des migrations à chaud reste entière pour les autres types d'applications transactionnelles. A notre connaissance, il n'existe pas de méthode de migrations à

56. https://camunda.com/
57. https://www.jbpm.org/
58. https://www.activiti.org/
59. https://www.ibm.com/support/knowledgecenter/fr/SSFPJS_8.5.6/com.ibm.wbpm.main.doc/topics/ibmbmp_overview.html
60. https://www.bizagi.com/

chaud pour applications transactionnelles multi-tenant. Comme nous l'avons vu, celles-ci sont importantes pour l'élasticité des applications transactionnelles s'appuyant sur des bases de données relationnelles. La méthode de migration à chaud que nous proposons dans la section 4.3.1 est basée sur des scripts SQL et la distribution des bases de données relationnelles correspondant à chaque ressource. Cette méthode, appliquée à un outil BPM, peut être généralisée à toute application Web transactionnelle multi-tenant. Il est envisageable de tester notre méthode sur différentes applications, et de la comparer à d'autres approches telles que Bucardo [61] ou Ghostferry [62]. Estimer le poids des migrations à chaud quelque soit la méthode employée est important pour l'élasticité. Une seconde étape serait de réaliser à l'aide de nos approches des tests plus intensifs sur le comportement des systèmes multi-tenants et des systèmes BPM, et d'étudier les migrations à chaud sur de plus grandes quantités de données, de tenants et de charges de travail. En effet, nous n'avons pas étudié les effets de la multitenancy, et avons considéré des tenants de même capacité pour les migrations et les injections. Il est également envisageable de faire évoluer notre module d'émulation de clients basé sur les systèmes multi-agents afin de pouvoir considérer des charges de travail plus réalistes.

Connaître les effets précis des migrations nous permettra d'avoir des méthodes d'allocation plus précises, permettant de respecter plus finement la qualité de service du client. Comme nous l'avons précisé dans les limites de notre approche, considérer le nombre de migrations reste une première approche. Nous avons pu voir dans le chapitre 4 que celles-ci ont des effets sur la qualité de service des tenants hébergés et colocalisés. L'algorithme d'allocation de ressources et de distribution des tenants devrait prendre en compte ces points. De plus, nous avons pris pour principe que, pour un pas de temps donné, un tenant doit être assigné à une seule ressource. Une évolution de notre algorithme pourrait permettre de distribuer sur plusieurs ressources différentes un même tenant. Cette répétition de données permettrait une tolérance aux pannes : une considération d'installations présentes sur plusieurs centres de données peut être envisagée. La latence et le coût du transport serait également à prendre en considération.

Nos méthodes d'allocation de ressources et de planification sont des méthodes hors-ligne (*offline*) nécessitant de connaître a priori les besoins des tenants en termes de débit de tâches. Le problème reste toutefois la gestion de besoins imprévus, qui pourraient amener les tenants à dépasser les besoins déclarés. Ceci pourrait avoir des effets négatifs sur les autres tenants hébergés. Une solution peut être de transformer notre solution en solution en ligne prenant en compte les besoins actuels et passés pour déterminer le futur, en plus des besoins déclarés. Une méthode de régression (à l'aide de méthodes d'apprentissage par exemple) peut être intéressante à employer. Des méthodes de process mining permettant de déduire les tâches employées par le processus peuvent également être intéressantes à employer.

Pour pouvoir utiliser nos méthodes en production, il est important de distribuer les requêtes HTTP des différents clients vers leur tenant et leur ressource respective. Ceci est habituellement fait à l'aide de proxy inverses. Or, les tenants peuvent être localisés sur différentes ressources selon le moment où la requête arrive. Le répartiteur de charge doit être couplé à l'algorithme appliquant nos méthodes afin que la requête arrive au bon endroit. Un référentiel contenant la correspondance entre les tenants et les ressources sur lesquelles ils sont hébergés doit être interprété par le proxy inverse pour déterminer à quelle ressource les requêtes doivent être envoyées. Il faut aussi prendre en compte les migrations : pendant les durées d'indisponibilité des tenants migrés, ces derniers ne sont pas accessibles. Les requêtes doivent être mises en attente pour être déclenchées ou renvoyées en erreur. Coupler nos méthodes à des outils du marché tels que l'outil Kubernetes [63] peut être intéressant pour démontrer leur utilisabilité dans des cas plus réalistes.

En s'appuyant sur ces concepts, nos méthodes d'allocation de ressources et de distribution de tenants permettront de réduire les coûts pour les fournisseurs, et ainsi la facture des utilisateurs finaux. Ces prochaines étapes aideront à la généralisation et à la simplification de ce type d'approches. Ces économies peuvent d'aider à l'adoption du BPM sur le cloud, ainsi que des offres SaaS en général.

61. `https://bucardo.org/Bucardo/`
62. `https://shopify.github.io/ghostferry/master/introduction.html`
63. `https://kubernetes.io/`

6.2.2 Élasticité des outils BPM

Pour les fournisseurs de BPMaaS, s'appuyer sur le débit de nombre de tâches comme mesure de qualité de service pour la mesure des ressources et tenants est une première étape intéressante. Toutefois, le comportement des outils BPM et les performances des exécutions de processus métiers n'est pas le même selon la nature des processus métiers [94]. Comme nous avons vu dans les expérimentations du chapitre 4, la différence de temps de réponse et des durées de migrations est significative selon les processus utilisés. Nous souhaitons répondre à la question suivante : y-a-t-il un moyen de déterminer les performances de l'exécution des processus métier en considérant plus précisément la structure des processus métiers ? Nous prévoyons de proposer une mesure faisant intervenir plusieurs éléments des processus (telles que le nombre de tâches, le nombre de branchements de chaque type, le nombre d'évènements, etc.) pour montrer qu'il est envisageable d'utiliser des mesures simples basées sur la structure des processus pour prédire leur temps de réponse. Des techniques d'apprentissage automatique peuvent être utiles dans ce but. Les tests préliminaires que nous avons effectué montrent qu'en s'appuyant sur des techniques de régression, on peut retrouver plus finement les temps d'exécution des processus en s'appuyant sur un ensemble de métriques globales liées au schéma.

Cette technique est intéressante et simple à mettre en œuvre dans le cadre d'approches hors-ligne, toutefois dans le cadre d'approches en ligne, une considération des chemins utilisés par le processus métiers peut être plus précise. L'utilisation de techniques de process mining [125], et plus précisément les méthodes liées à la prédiction peut être avantageuses pour déterminer les tâches effectuées dans le futur et obtenir une meilleure estimation des capacités nécessaires à l'exécution des processus. L'utilisation d'approches prédictives décrite dans la sous-section précédente peut être également élargie en utilisant des approches de process mining : déterminer l'ordonnancement des processus (en représentant ce qu'on pourrait nommer des méta processus), permettait d'estimer les capacités nécessaires plus finement. Ceci passerait par la prédiction des processus qui seront déclenchés dans le futur, et ainsi une meilleure adaptation des ressources à leurs besoins.

L'exécution de tâches humaines, de par leur nature, est totalement liée au comportement des utilisateurs du système. La considération des performances des tâches humaines est un sujet peu étudié, ce qui est lié au manque de simulation pour ces derniers. Le comportement des utilisateurs sera spécifique selon le rôle de l'utilisateur, sa méthode de travail, sa charge de travail. Pour cela, représenter finement leur comportement peut aider à une meilleure estimation des capacités. Notre approche de simulation d'utilisateur basée sur les systèmes multi-agents (présentée dans le chapitre 4) représente une première étape assurant pour l'ensemble des clients émulés le même comportement (récupération, assignation et exécution des tâches immédiate), mais celle-ci peut être élargie et fournir une simulation plus réaliste. On peut envisager différentes durées d'attente, la prise en considération des variables. Les approches de process mining ou de Robotic Process Automation (RPA) peuvent être utiles dans ce cadre.

6.2.3 Micro services et Edge Computing

Nous avons considéré dans nos approches des moteurs BPM à architecture monolithique. Une alternative existe : les architectures micro-services, dont le principe est de séparer les applications en multiples modules indépendants communiquant habituellement par HTTP. Ces applications ont une meilleure scalabilité que les architectures monolothiques [126], et, dans ce type d'applications, chaque service a sa propre base de données ceci retirant la problématique du passage à l'échelle de la couche persistance [126]. La considération des opérations de migrations à chaud, en plus des opérations d'élasticité est un sujet important [127]. Ce type d'architecture est également envisagé pour les outils BPM [128].

Les architectures micro-services sont parfois également utilisées dans le contexte récent de l'edge computing [129]. Le principe ici est de profiter de la puissance des smartphones pour effectuer les calculs dans le cadre de distances (et donc de latences) élevées entre les applications des utilisateurs finaux et l'infrastructure du cloud, comme par exemple pour la reconnaissance d'objets. Les techniques de migration à chaud sont également un point important, les applications étant encapsulées dans des machines virtuelles ou des conteneurs, même si la migration de ces derniers n'a été que très peu étudiée [130]. Des versions améliorées de nos algorithmes, frameworks et techniques de migrations peuvent être utiles dans ce cadre. Des problèmes tels que la latence avec les centres de données, l'absence de connectivité ou la puissance

limitée des terminaux sont à considérer.

Annexe A

Détails sur le framework d'estimation de ressources

A.1 Détails sur le framework d'estimation de ressources

A.1.1 Ansible

Ansible s'appuie sur les concepts suivants :

— hôtes et groupes : les hôtes représentent les instances ciblées et leurs types, identifiables par un nom de domaine ou une adresse IP. Un groupe représente un sous-ensemble d'hôtes ayant une fonction précise (par exemple : bases de données, serveur d'application, etc.).

— inventaire : liste d'hôtes et de groupes. Un lancement de l'outil concerne un inventaire. Un inventaire peut être statique (un fichier décrivant la liste des hôtes, groupes et variables associées), ou dynamique (dans ce cas un script se charge de renvoyer les valeurs nécessaires).

— tâche : un appel à un module, ou à un script. De nombreux modules sont fournis en standard (ceux-ci peuvent concerner par exemple le cloud, le monitoring, etc.). Il est également possible de coder ses propres modules.

— rôle : un ensemble de tâches réutilisable. Le but d'un rôle peut être par exemple l'installation d'un composant.

— playbook : un script faisant appel aux rôles et groupes. Un inventaire sera lié à l'exécution afin de réaliser les opérations le composant.

— variables : des variables de plusieurs types peuvent être déclarées et utilisées dans la plupart des concepts décrits (par exemple une variable décrite au niveau d'un playbook, d'un rôle, d'un hôte, etc.). Les variables peuvent être également surchargées à l'exécution du script.

— templates : afin de générer les fichiers de configuration nécessaires, il est possible de préparer des templates Jinja2. Les variables définies dans ces derniers seront remplacées par les valeurs correspondantes des variables à l'exécution.

Nous avons défini plusieurs rôles et playbooks pour nos besoins, tels que des rôles pour la partie base de données, injecteur, etc.

A.1.2 Jenkins

Les tâches à réaliser sont entrées sous forme de *build jobs*, appelés plus couramment *jobs*. Un job consiste en une phase d'initialisation, de récupération du code testé via un gestionnaire de code (CVS), et ensuite de différentes étapes correspondant à l'exécution de tâches (*build step*). Il est possible de définir différents types d'étapes (exécution de script Shell, exécution de tâche Maven, etc.).

Jenkins ajoute à cela le concept de *pipelines*, se situant au même niveau que les jobs. Un pipeline permet d'orchestrer plusieurs jobs et est défini dans un langage de script basé sur Groovy.

Pipelines et *jobs* peuvent prendre des paramètres en entrée, que l'utilisateur peut choisir à l'aide de listes déroulantes. Ce dernier point nous a servi à fixer les valeurs des paramètres du test, tels que le

nombre de tests, la version du BPMS à tester, le type d'instance pour la base de données, etc.

Un aspect intéressant des pipelines est la puissance du langage de script proposé. Contrairement aux jobs, que l'on définit par une série de tâches avec peu de contrôle sur le flot de contrôle, il est possible d'ajouter des traitements d'exceptions au niveau d'un pipeline. Ceci nous a en particulier permis de gérer des cas non prévus, et plus particulièrement d'assurer la libération des ressources du cloud, même si une erreur est survenue dans un des traitements intermédiaires. En effet, un des aspects importants est le coût des ressources, et la libération des ressources dans tous les cas de figures est primordiale. Ce point est important pour l'aspect *économique* du framework.

Afin de nous aider dans notre tâche nous avons ajouté plusieurs extensions à nos installations, plus précisément :

- le plug-in Ansible, permettant une exécution simplifiée des tâches Ansible en fournissant divers menus.
- le plug-in Rebuilder, permettant de relancer un build paramétré (fonctionnalité non présente par défaut). Ceci nous permet en l'occurrence de relancer des tests sans avoir à entrer à nouveau tous les paramètres.
- le plug-in Environment injector, permettant d'initialiser des variables d'environnement dans le contexte des jobs.
- le plug-in Conditional Buildstep, permettant d'exécuter des build step de manière conditionnelle, basé sur des variables en entrée.

Bibliographie

[1] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12) :1281–1294, November 2008.

[2] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Elasticity in cloud computing : State of the art and research challenges. *IEEE Transactions on Services Computing*, 2017.

[3] George Feuerlicht and Jaroslav Pokorný. Can Relational DBMS Scale Up to the Cloud ? In Rob Pooley, Jennifer Coady, Christoph Schneider, Henry Linger, Chris Barry, and Michael Lang, editors, *Information Systems Development*, pages 317–328. Springer New York, 2013.

[4] Philipp Leitner and Jürgen Cito. Patterns in the Chaos—A Study of Performance Variation and Predictability in Public IaaS Clouds. *ACM Transactions on Internet Technology*, 16(3) :1–23, April 2016.

[5] J. Emeras, S. Varrette, and P. Bouvry. Amazon Elastic Compute Cloud (EC2) vs. In-House HPC Platform : A Cost Analysis. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 284–293, June 2016.

[6] Fabian Brosig, Nikolaus Huber, and Samuel Kounev. Architecture-level software performance abstractions for online performance prediction. *Science of Computer Programming*, 90 :71–92, September 2014.

[7] Terence Kelly and Alex Zhang. Predicting performance in distributed enterprise applications. *HP Laboratories Technical Report HPL-2006-76*, 2006.

[8] Andrei Solomon and Marin Litoiu. Business process performance prediction on a tracked simulation model. In *Proceeding of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems - PESOS '11*, pages 50–56, Waikiki, Honolulu, HI, USA, 2011. ACM Press.

[9] Alan T Litchfield and Jacqui Althouse. A Systematic Review of Cloud Computing, Big Data and Databases on the Cloud. *Service Systems*, page 20, 2014.

[10] Sudipto Das, Shoji Nishimura, Divyakant Agrawal, and Amr El Abbadi. Live database migration for elasticity in a multitenant database for cloud platforms. Technical report, CS, UCSB, Santa Barbara, CA, USA, 2010.

[11] G. Rosinosky, S. Youcef, and F. Charoy. A Framework for BPMS Performance and Cost Evaluation on the Cloud. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 653–658, December 2016.

[12] Guillaume Rosinosky, Chahrazed Labba, Vincenzo Ferme, Samir Youcef, François Charoy, and Cesare Pautasso. Evaluating Multi-Tenant Live Migrations Effects on Performance. In *26th International Conference on COOPERATIVE INFORMATION SYSTEMS (CoopIS)*, Valetta, Malta, October 2018.

[13] G. Rosinosky, S. Youcef, and F. Charoy. An Efficient Approach for Multi-tenant Elastic Business Processes Management in Cloud Computing Environment. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 311–318, June 2016.

[14] Guillaume Rosinosky, Samir Youcef, and François Charoy. Efficient Migration-Aware Algorithms for Elastic BPMaaS. In Josep Carmona, Gregor Engels, and Akhil Kumar, editors, *Business Process Management*, volume 10445, pages 147–163. Springer International Publishing, Cham, 2017.

[15] Guillaume Rosinosky, Samir Youcef, and Francois Charoy. A genetic algorithm for cost-aware business processes execution in the cloud. In *ICSOC 2018 - The 16th International Conference on Service-Oriented Computing*, Hangzhou, China, November 2018.

[16] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A View of Cloud Computing. *Commun. ACM*, 53(4) :50–58, April 2010.

[17] Lloyd G. Williams and Connie U. Smith. Web Application Scalability : A Model-Based Approach. In *30th International Computer Measurement Group Conference, December 5-10, 2004, Las Vegas, Nevada, USA, Proceedings*, pages 215–226. Computer Measurement Group, 2004.

[18] Zhou Wei, Guillaume Pierre, and Chi-Hung Chi. CloudTPS : Scalable Transactions for Web Applications in the Cloud. *IEEE Transactions on Services Computing*, 5(4) :525–539, 24.

[19] ISO 9000 :2015(fr), Systèmes de management de la qualité — Principes essentiels et vocabulaire. https ://www.iso.org/obp/ui/#iso :std :iso :9000 :ed-4 :v2 :fr.

[20] BPMN Specification - Business Process Model and Notation. http ://www.bpmn.org/.

[21] Cloud app management – from migration to monitoring. http ://searchcloudcomputing.techtarget.com/feature/Cloud-app-management-from-migration-to-monitoring.

[22] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. Technical Report NIST Special Publication (SP) 800-145, National Institute of Standards and Technology, September 2011.

[23] Rui Esteves. A taxonomic analysis of cloud computing. In *1st Doctoral Workshop in Complexity Sciences ISCTE-IUL/FCUL*, 2011.

[24] Daniel Paschek, Adelin Trusculescu, Adrian Mateescu, and Anca Draghici. Business Process as a Service - a Flexible Approach for it Service Management and Business Process Outsourcing. In *Management Challenges in a Network Economy : Proceedings of the MakeLearn and TIIM International Conference 2017*, pages 195–203. ToKnowPress, 2017.

[25] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in Cloud Computing : What It Is, and What It Is Not. In *ICAC*, pages 23–27, 2013.

[26] Yousri Kouki, Thomas Ledoux, Damián Serrano, Sara Bouchenak, Jonathan Lejeune, Luciana Arantes, Julien Sopena, Pierre Sens, and Paris LIP6–INRIA. SLA et qualité de service pour le Cloud Computing. In *Conférence d'informatique En Parallélisme, Architecture et Système, ComPAS 2013*, pages 1–11, 2013.

[27] Sebastian Lehrig, Hendrik Eikerling, and Steffen Becker. Scalability, Elasticity, and Efficiency in Cloud Computing : A Systematic Literature Review of Definitions and Metrics. In *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, QoSA '15, pages 83–92, New York, NY, USA, 2015. ACM.

[28] Kai Hwang, Yue Shi, and Xiaoying Bai. Scale-Out vs. Scale-Up Techniques for Cloud Performance and Productivity. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 763–768, Singapore, Singapore, December 2014. IEEE.

[29] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing*, 12(4) :559–592, December 2014.

[30] A. Evangelidis, D. Parker, and R. Bahsoon. Performance Modelling and Verification of Cloud-Based Auto-Scaling Policies. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 355–364, May 2017.

[31] Andreas Wolke, Boldbaatar Tsend-Ayush, Carl Pfeiffer, and Martin Bichler. More than bin packing : Dynamic resource allocation strategies in cloud data centers. *Information Systems*, 52 :83–95, August 2015.

[32] Yusen Li, Xueyan Tang, and Wentong Cai. On Dynamic Bin Packing for Resource Allocation in the Cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pages 2–11, New York, NY, USA, 2014. ACM.

[33] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[34] R. Righi, V. Rodrigues, C. Andre daCosta, G. Galante, L. Bona, and T. Ferreto. AutoElastic : Automatic Resource Elasticity for High Performance Applications in the Cloud. *IEEE Transactions on Cloud Computing*, PP(99) :1–1, 2015.

[35] Kaveh Razavi, Ana Ion, Genc Tato, Kyuho Jeong, Renato Figueiredo, Guillaume Pierre, and Thilo Kielmann. Kangaroo : A Tenant-Centric Software-Defined Cloud Infrastructure. In *Proceedings of the IEEE International Conference on Cloud Engineering*, 2015.

[36] Challenge ROADEF/EURO 2012 : Machine Reassignment. http ://challenge.roadef.org/2012/en/sujet.php.

[37] Haris Gavranović, Mirsad Buljubašić, and Emir Demirović. Variable Neighborhood Search for Google Machine Reassignment problem. *Electronic Notes in Discrete Mathematics*, 39 :209–216, December 2012.

[38] W. Jaśkowski, M. Szubert, and P. Gawron. A hybrid MIP-based large neighborhood search heuristic for solving the machine reassignment problem. *Annals of Operations Research*, January 2015.

[39] Felix Brandt, Jochen Speck, and Markus Völker. Constraint-based large neighborhood search for machine reassignment : A solution approach to the ROADEF/EURO challenge 2012. *Annals of Operations Research*, December 2014.

[40] J. Schanffner, Dean Jacobs, Tim Kraska, and Hasso Plattner. The Multi-Tenant Data Placement Problem. *Proc. DBKDA*, pages 157–162, 2012.

[41] Wei-Tek Tsai, Yu Huang, Xiaoying Bai, and J. Gao. Scalable Architectures for SaaS. In *2012 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, pages 112–117, April 2012.

[42] Michael Stonebraker. New Opportunities For New SQL. https ://cacm.acm.org/magazines/2012/11/156577-new-opportunities-for-new-sql/fulltext, November 2012.

[43] Ricardo Jimenez-Peris, Marta Patino-Martinez, Bettina Kemme, Ivan Brondino, José Pereira, Ricardo Vilaça, Francisco Cruz, Rui Oliveira, and Yousuf Ahmad. CumuloNimbo : A Cloud Scalable Multi-tier SQL Database. *Data Engineering*, page 73, 2015.

[44] Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, and others. F1 : A distributed SQL database that scales. *Proceedings of the VLDB Endowment*, 6(11) :1068–1079, 2013.

[45] Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. ElasTraS : An elastic, scalable, and self-managing transactional database for the cloud. *ACM Transactions on Database Systems*, 38(1) :1–45, April 2013.

[46] Aaron J. Elmore, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Zephyr : Live migration in shared nothing databases for elastic cloud platforms. In *Proceedings of the 2011 International Conference on Management of Data - SIGMOD '11*, page 301, Athens, Greece, 2011. ACM Press.

[47] Olumuyiwa Oluwafunto Matthew, Carl Dudley, and Robert Moreton. A review of multi-tenant database and factors that influence its adoption. In *UK Academy for Information Systems Conference Proceedings 2014*, page 22. Association for Information Systems, 2014.

[48] Database Live-Migration with Oracle Multitenant and the Oracle Universal Connection Pool (UCP) on Oracle Real Application Clusters (RAC) - pdblivemigration-2301324.pdf. http ://www.oracle.com/technetwork/database/multitenant/learn-more/pdblivemigration-2301324.pdf.

[49] Ziyang Liu, Hakan Hacıgümüş, Hyun Jin Moon, Yun Chi, and Wang-Pin Hsiung. PMAX : Tenant placement in multitenant databases for profit maximization. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 442–453. ACM, 2013.

[50] Sean Kenneth Barker, Yun Chi, Hakan Hacigümüs, Prashant J. Shenoy, and Emmanuel Cecchet. ShuttleDB : Database-Aware Elasticity in the Cloud. In *11th International Conference on Autonomic Computing, ICAC '14, Philadelphia, PA, USA, June 18-20, 2014.*, pages 33–43, 2014.

[51] Jan Schaffner, Tim Januschowski, Megan Kercher, Tim Kraska, Hasso Plattner, Michael J. Franklin, and Dean Jacobs. RTP : Robust tenant placement for elastic in-memory database clusters. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 773–784. ACM, 2013.

[52] Marco Serafini, Essam Mansour, Ashraf Aboulnaga, Kenneth Salem, Taha Rafiq, and Umar Farooq Minhas. Accordion : Elastic scalability for database systems supporting distributed transactions. *Proceedings of the VLDB Endowment*, 7(12) :1035–1046, 2014.

[53] Santiago Gómez Sáez, Vasilios Andrikopoulos, Marina Bitsaki, Frank Leymann, and André Van Hoorn. Utility-Based Decision Making for Migrating Cloud-Based Applications. *ACM Transactions on Internet Technology*, 18(2) :1–22, February 2018.

[54] Manish Verma, G. R. Gangadharan, Nanjangud C. Narendra, Ravi Vadlamani, Vidyadhar Inamdar, Lakshmi Ramachandran, Rodrigo N. Calheiros, and Rajkumar Buyya. Dynamic resource demand prediction and allocation in multi-tenant service clouds : Dynamic Resource Demand Prediction and Allocation in Multi-tenant Service Clouds. *Concurrency and Computation : Practice and Experience*, 28(17) :4429–4442, December 2016.

[55] Enfeng Yang, Yong Zhang, Lei Wu, Yulong Liu, and Shijun Liu. A Hybrid Approach to Placement of Tenants for Service-Based Multi-tenant SaaS Application. In *Services Computing Conference (APSCC), 2011 IEEE Asia-Pacific*, pages 124–130, December 2011.

[56] Yongqing Zheng, Xiaojun Ren, and Lanju Kong. Multi-tenant Data Migration Strategy in SaaS Platform. *Research Journal of Applied Sciences, Engineering and Technology*, 7(12) :2421–2426, March 2014.

[57] Christoph Fehling, Frank Leymann, and Ralph Mietzner. A framework for optimized distribution of tenants in cloud applications. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference On*, pages 252–259. IEEE, 2010.

[58] G. B. Pallavi and P. Jayarekha. A Proposed Resource Sharing Architecture for Multitenant SaaS Applications. In Suresh Chandra Satapathy, Vikrant Bhateja, and Amit Joshi, editors, *Proceedings of the International Conference on Data Engineering and Communication Technology*, volume 468, pages 19–26. Springer Singapore, Singapore, 2017.

[59] Javier Espadas, Arturo Molina, Guillermo Jiménez, Martín Molina, Raúl Ramírez, and David Concha. A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures. *Future Generation Computer Systems*, 29(1) :273–286, January 2013.

[60] Xianzhang Chen and Xiaoping Li. A Dynamic Resource Balance Algorithm for Multi-tenant Placement Problem in SaaS. In *2013 International Conference on Service Sciences (ICSS)*, pages 123–128, April 2013.

[61] Bhaskar Prasad Rimal and Martin Maier. Workflow Scheduling in Multi-Tenant Cloud Computing Environments. *IEEE Transactions on Parallel and Distributed Systems*, 28(1) :290–304, January 2017.

[62] Sumit Kalra and T. V. Prabhakar. Towards Dynamic Tenant Management for Microservice Based Multi-Tenant SaaS Applications. In *Proceedings of the 11th Innovations in Software Engineering Conference*, ISEC '18, pages 12 :1–12 :5, New York, NY, USA, 2018. ACM.

[63] Stefan Schulte, Christian Janiesch, Srikumar Venugopal, Ingo Weber, and Philipp Hoenisch. Elastic Business Process Management : State of the art and open challenges for BPM in the cloud. *Future Generation Computer Systems*, 2014.

[64] Jiajie Xu, Chengfei Liu, Xiaohui Zhao, Sira Yongchareon, and Zhiming Ding. Resource Management for Business Process Scheduling in the Presence of Availability Constraints. *ACM Transactions on Management Information Systems*, 7(3) :1–26, October 2016.

[65] Almir Djedović, Emir Žunić, Zikrija Avdagić, and Almir Karabegović. Optimization of business processes by automatic reallocation of resources using the genetic algorithm. In *Telecommunications (BIHTel), 2016 XI International Symposium On*, pages 1–7. IEEE, 2016.

[66] P. Fernández, H. L. Truong, S. Dustdar, and A. Ruiz-Cortés. Programming Elasticity and Commitment in Dynamic Processes. *IEEE Internet Computing*, 19(2) :68–74, March 2015.

[67] W. Sellami, H. H. Kacem, and A. H. Kacem. Elastic Multi-tenant Business Process Based Service Pattern in Cloud Computing. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 154–161, December 2014.

[68] Mohamed Mohamed, Mourad Amziani, Djamel Belaïd, Samir Tata, and Tarek Melliti. An autonomic approach to manage elasticity of business processes in the Cloud. *Future Generation Computer Systems*, 50 :49–61, September 2015.

[69] Philipp Hoenisch, Dieter Schuller, Stefan Schulte, Christoph Hochreiner, and Schahram Dustdar. Optimization of Complex Elastic Processes. *IEEE Transactions on Services Computing*, 9(5) :700–713, September 2016.

[70] Philipp Hoenisch. ViePEP - A BPMS for Elastic Processes. In *ZEUS*, 2014.

[71] Philipp Hoenisch, Christoph Hochreiner, Dieter Schuller, Stefan Schulte, Jan Mendling, and Schahram Dustdar. Cost-Efficient Scheduling of Elastic Processes in Hybrid Clouds. In *Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing*, CLOUD '15, pages 17–24, Washington, DC, USA, 2015. IEEE Computer Society.

[72] Olena Skarlat, Philipp Hoenisch, and Schahram Dustdar. On Energy Efficiency of BPM Enactment in the Cloud. In Manfred Reichert and Hajo A. Reijers, editors, *Business Process Management Workshops*, volume 256, pages 489–500. Springer International Publishing, Cham, 2016.

[73] P. Hoenisch, S. Schulte, S. Dustdar, and S. Venugopal. Self-Adaptive Resource Allocation for Elastic Process Execution. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 220–227, June 2013.

[74] S. Euting, C. Janiesch, R. Fischer, S. Tai, and I. Weber. Scalable Business Process Execution in the Cloud. In *2014 IEEE International Conference on Cloud Engineering (IC2E)*, pages 175–184, March 2014.

[75] C. Janiesch, I. Weber, J. Kuhlenkamp, and M. Menzel. Optimizing the Performance of Automated Business Processes Executed on Virtualized Infrastructure. In *2014 47th Hawaii International Conference on System Sciences*, pages 3818–3826, January 2014.

[76] Mourad Amziani. *Modeling, Evaluation and Provisioning of Elastic Service-Based Business Processes in the Cloud*. Thesis, Evry, Institut national des télécommunications, June 2015.

[77] A. B. Jrada, S. Bhiria, and S. Tata. Data-Aware Modeling of Elastic Processes for Elasticity Strategies Evaluation. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 464–471, June 2017.

[78] M. Rekik, K. Boukadi, N. Assy, W. Gaaloul, and H. Ben-Abdallah. A Linear Program for Optimal Configurable Business Processes Deployment into Cloud Federation. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 34–41, June 2016.

[79] R. B. Halima, S. Kallel, W. Gaaloul, and M. Jmaiel. Optimal Cost for Time-Aware Cloud Resource Allocation in Business Process. In *2017 IEEE International Conference on Services Computing (SCC)*, pages 314–321, June 2017.

[80] Emna Hachicha, Nour Assy, Walid Gaaloul, and Jan Mendling. A configurable resource allocation for multi-tenant process development in the cloud. In *International Conference on Advanced Information Systems Engineering*, pages 558–574. Springer, 2016.

[81] E. Hachicha, K. Yongsiriwit, M. Sellami, and W. Gaaloul. Genetic-Based Configurable Cloud Resource Allocation in QoS-Aware Business Process Development. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 836–839, June 2017.

[82] M. Ahmed-Nacer, K. Suri, M. Sellami, and W. Gaaloul. Simulation of Configurable Resource Allocation for Cloud-Based Business Processes. In *2017 IEEE International Conference on Services Computing (SCC)*, pages 305–313, June 2017.

[83] R. Ben Halima, I. Zouaghi, S. Kallel, W. Gaaloul, and M. Jmaiel. Formal Verification of Temporal Constraints and Allocated Cloud Resources in Business Processes. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 952–959, May 2018.

[84] M. Graiet, A. Mammar, S. Boubaker, and W. Gaaloul. Towards Correct Cloud Resource Allocation in Business Processes. *IEEE Transactions on Services Computing*, 10(1) :23–36, January 2017.

[85] Toni Mastelic, Walid Fdhila, Ivona Brandic, and Stefanie Rinderle-Ma. Predicting Resource Allocation and Costs for Business Processes in the Cloud. In *Services (SERVICES), 2015 IEEE World Congress On*, pages 47–54. IEEE, 2015.

[86] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan. Business Process Scheduling Strategies in Cloud Environments with Fairness Metrics. In *2013 IEEE International Conference on Services Computing*, pages 519–526, June 2013.

[87] Kahina Bessai, Samir Youcef, Ammar Oulamara, Claude Godart, and Selmin Nurcan. Bi-criteria workflow tasks allocation and scheduling in Cloud computing environments. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference On*, pages 638–645. IEEE, 2012.

[88] Eun-Kyu Byun, Yang-Suk Kee, Jin-Soo Kim, and Seungryoul Maeng. Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, 27(8) :1011–1026, October 2011.

[89] Saurabh Bilgaiyan, Santwana Sagnika, and Madhabananda Das. An Analysis of Task Scheduling in Cloud Computing using Evolutionary and Swarm-based Algorithms. *International Journal of Computer Applications*, 89(2), 2014.

[90] François Charoy, Adnene Guabtni, and Miguel Valdes Faura. A dynamic workflow management system for coordination of cooperative activities. In *Business Process Management Workshops*, pages 205–216. Springer, 2006.

[91] Multitenancy - Gartner IT Glossary. http ://www.gartner.com/it-glossary/multitenancy.

[92] Vladislav Lazarov. Comparison of Different Implementations of Multi-Tenant Databases. BACHELORARBEIT, TUM Computer Science, TUM Computer Science, July 2007.

[93] Sherif Sakr. Cloud-hosted databases : Technologies, challenges and opportunities. *Cluster Computing*, 17(2) :487–502, June 2014.

[94] Vincenzo Ferme, Ana Ivanchikj, and Cesare Pautasso. Estimating the Cost for Executing Business Processes in the Cloud. In *Business Process Management Forum*, Lecture Notes in Business Information Processing, pages 72–88. Springer, Cham, September 2016.

[95] Daniel A. Menascé. Load testing of web sites. *IEEE Internet Computing*, 6(4) :70–74, 2002.

[96] Karl Huppler. The Art of Building a Good Benchmark. In Raghunath Nambiar and Meikel Poess, editors, *Performance Evaluation and Benchmarking*, Lecture Notes in Computer Science, pages 18–30. Springer Berlin Heidelberg, 2009.

[97] Vincenzo Ferme, Ana Ivanchikj, and Cesare Pautasso. A framework for benchmarking BPMN 2.0 workflow management systems. In *International Conference on Business Process Management*, pages 251–259. Springer, 2015.

[98] Carl Boettiger. An introduction to Docker for reproducible research, with examples from the R environment. *arXiv preprint arXiv :1410.0846*, 2014.

[99] Paul Venezia. Review : Puppet vs. Chef vs. Ansible vs. Salt. http ://www.infoworld.com/article/2609482/data-center/data-center-review-puppet-vs-chef-vs-ansible-vs-salt.html, 2013-11-21T06 :00-05 :00.

[100] Jim Melton, Jan Eike Michels, Vanja Josifovski, Krishna Kulkarni, and Peter Schwarz. SQL/MED : A Status Report. *SIGMOD Rec.*, 31(3) :81–89, September 2002.

[101] Aziz Murtazaev and Sangyoon Oh. Sercon : Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing. *IETE Technical Review*, 28(3) :212, 2011.

[102] Sean Barker, Yun Chi, Hyun Jin Moon, Hakan Hacigümüş, and Prashant Shenoy. Cut me some slack : Latency-aware live migration for databases. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 432–443. ACM, 2012.

[103] Ozgun Ali Erdogan, Sumedh Suhas Pathak, and Hadi Moshayedi. Apparatus and Method for Operating a Distributed Database with Foreign Tables, March 2014.

[104] Xuefei Wang, Ruohang Feng, Wei Dong, Xiaoqian Zhu, and Wenke Wang. Unified Access Layer with PostgreSQL FDW for Heterogeneous Databases. In Xuanhua Shi, Hong An, Chao Wang, Mahmut Kandemir, and Hai Jin, editors, *Network and Parallel Computing*, Lecture Notes in Computer Science, pages 131–135. Springer International Publishing, 2017.

[105] Jacques Ferber. *Multi-Agent Systems : An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, Harlow, 1999.

[106] Tobias Küster, Marco Lützenberger, Axel Heßler, and Benjamin Hirsch. Integrating process modelling into multi-agent system engineering. *Multiagent and Grid Systems*, 8(1) :105–124, January 2012.

[107] Hoa Khanh Dam, Aditya Ghose, and Mohammad Qasim. An Agent-Mediated Platform for Business Processes. *International Journal of Information Technology and Web Engineering (IJITWE)*, 10(2) :43–61, April 2015.

[108] Nguyen Tuan Thanh Le, Chihab Hanachi, Serge Stinckwich, and Tuong Vinh Ho. Mapping BPMN Processes to Organization Centered Multi-Agent Systems to Help Assess Crisis Models. In Manuel Núñez, Ngoc Thanh Nguyen, David Camacho, and Bogdan Trawiński, editors, *Computational Collective Intelligence*, Lecture Notes in Computer Science, pages 77–88. Springer International Publishing, 2015.

[109] Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling : A cross-disciplinary survey. *Constraints*, 17(1) :51–85, January 2012.

[110] Konstantinos Koiliaris and Chao Xu. A Faster Pseudopolynomial Time Algorithm for Subset Sum. *arXiv preprint arXiv :1507.02318*, 2015.

[111] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer Science & Business Media, February 2004.

[112] Subset sum problem - Rosetta Code. http ://rosettacode.org/wiki/Subset_sum_problem#Brute_force.

[113] Mauro Maria Baldi, Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei. Branch-and-price and beam search algorithms for the Variable Cost and Size Bin Packing Problem with optional items. *Annals of Operations Research*, 222(1) :125–141, November 2014.

[114] Mohamed Haouari and Mehdi Serairi. Relaxations and exact solution of the variable sized bin packing problem. *Computational Optimization and Applications*, 48(2) :345–368, March 2011.

[115] Mohamed Maiza, Abdenour Labed, and Mohammed Said Radjef. Efficient algorithms for the offline variable sized bin-packing problem. *Journal of Global Optimization*, 57(3) :1025–1038, November 2013.

[116] Vera Hemmelmayr, Verena Schmid, and Christian Blum. Variable neighbourhood search for the variable sized bin packing problem. *Computers & Operations Research*, 39(5) :1097–1108, May 2012.

[117] Jangha Kang and Sungsoo Park. Algorithms for the variable sized bin packing problem. *European Journal of Operational Research*, 147(2) :365–372, 2003.

[118] Hashem A. Isa, Saleh Oqeili, and Sulieman Bani-Ahmad. The Subset-Sum Problem : Revisited with an Improved Approximated Solution. *International Journal of Computer Applications*, 114(14) :1–5, 2015.

[119] Miodrag Lovrić, Marina Milanović, and Milan Stamenković. Algoritmic methods for segmentation of time series : An overview. *Journal of Contemporary Economic and Business Issues*, 1(1) :31–53, 2014.

[120] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference On*, pages 289–296. IEEE, 2001.

[121] Carl Sandrock. Identification and Generation of Realistic Input Sequences for Stochastic Simulation with Markov Processes. In Shkelzen Cakaj, editor, *Modeling Simulation and Optimization - Tolerance and Optimal Control*. InTech, April 2010.

[122] Aaron Garrett. Inspyred : Bio-inspired Algorithms in Python — inspyred 1.0 documentation. http ://pythonhosted.org/inspyred/, 2014.

[123] Stuart Mitchell, Michael OSullivan, and Iain Dunning. PuLP : A linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand, http ://www. optimization-online. org/DB_ FILE/2011/09/3178. pdf*, 2011.

[124] Cor-Paul Bezemer and Andy Zaidman. Multi-tenant SaaS Applications : Maintenance Dream or Nightmare ? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, IWPSE-EVOL '10, pages 88–92, New York, NY, USA, 2010. ACM.

[125] Wil van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos Buijs, Andrea Burattin, Josep Carmona, Malu Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian Günther, Antonella Guzzo, Paul Harmon, Arthur ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maggi, Donato Malerba, Ronny S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari-Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Wynn. Process Mining Manifesto. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops*, Lecture Notes in Business Information Processing, pages 169–194. Springer Berlin Heidelberg, 2012.

[126] Wilhelm Hasselbring. Microservices for Scalability : Keynote Talk Abstract. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering - ICPE '16*, pages 133–134, Delft, The Netherlands, 2016. ACM Press.

[127] Maria Fazio, Antonio Celesti, Rajiv Ranjan, Chang Liu, Lydia Chen, and Massimo Villari. Open Issues in Scheduling Microservices in the Cloud. *IEEE Cloud Computing*, 3(5) :81–88, September 2016.

[128] Sascha Alpers, Christoph Becker, Andreas Oberweis, and Thomas Schuster. Microservice Based Tool Support for Business Process Modelling. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, pages 71–78, Adelaide, Australia, September 2015. IEEE.

[129] Dmitry Namiot and Manfred Sneps-Sneppe. On Micro-services Architecture. *International Journal of Open Information Technologies*, 2(9) :24–27, August 2014.

[130] Andrew Machen, Shiqiang Wang, Kin K. Leung, Bong Jun Ko, and Theodoros Salonidis. Live Service Migration in Mobile Edge Clouds. *IEEE Wireless Communications*, 25(1) :140–147, February 2018.

Résumé

La disponibilité de plateformes middleware dans le cloud, avec un passage à l'échelle transparent est un vrai progrès pour les développeurs et les intégrateurs logiciels. Ils peuvent développer et déployer leurs applications sans s'inquiéter des détails opérationnels. Cependant, le coût d'exploitation d'une infrastructure dans le cloud peut devenir rapidement important. Les fournisseurs doivent disposer de méthodes pour le réduire en adaptant la taille des ressources aux besoins des clients. Dans cette thèse, nous nous focalisons sur les applications Web multi-tenant transactionnelles, plus particulièrement les moteurs d'exécution de processus métiers. Nous proposons des méthodes permettant d'optimiser les coûts opérationnels d'un fournisseur d'exécution de processus "en tant que service" (BPMaaS) tout en assurant un niveau suffisant de qualité de service. Ce type d'applications ne passe pas facilement à l'échelle à cause de sa couche persistance et de la nature transactionnelle des opérations. Il faut distribuer les installations des clients de manière à optimiser les coûts et éventuellement les déplacer en fonction de l'évolution de la charge. Ces déplacements (ou migrations) ont un impact sur la qualité de service et il faut les limiter. Dans un premier temps, nous proposons une méthode de mesure de la capacité des ressources du cloud en termes de débit d'exécution de tâches BPM, puis nous proposons une méthode de mesure de l'impact des migrations que nous avons évalué, ceci confirmant nos hypothèses. Ensuite, nous proposons plusieurs modèles d'optimisation linéaire, ainsi que des heuristiques d'allocation de ressources et de distribution des clients prenant en compte le coût de l'infrastructure, la capacité des ressources et les besoins des clients, tout en limitant les nombres de migrations. Ces modèles sont fondés sur la connaissance de l'évolution de la charge des clients par unité de temps. Nous avons expérimenté les trois méthodes que nous avons proposées sur la solution BPM Bonita, et montré qu'elles permettent des gains substantiels sur l'exploitation de l'infrastructure par rapport à une méthode basique.

Mots-clés: élasticité, BPM, cloud, migration à chaud

Abstract

The availability of middleware platforms in the cloud, with "transparent" scalability, is a progress for software developers and integrators. They can develop and deploy their applications without worrying about technical details. However, the exploitation cost of a cloud infrastructure can quickly become important. Providers requires methods to reduce this cost by adapting the size of ressources to the needs of the customers. In this thesis, we focus on multi-tenant transactional web applications, more precisely on business processes execution engines. We propose methods allowing to optimize the operational costs of providers of business process execution "as a Service" (BPMaaS) while ensuring a sufficient level of quality of service. This type of application do not scale well because of its persistence tier and of the transactional nature of operations. One must distribute the customers installations in order to optimize the cost, and sometimes move them depending of the needs of the customers. These moves (or migrations) have an impact on the quality of service and they must be limited. First, we propose a method for measuring the size of resources in terms of BPM tasks throughput, and then a method for measuring the impact of migrations we evaluate, thus confirming our hypothesis. We also propose several linear optimization models and heuristics targeting resouce allocation and distribution of customers, while limiting the number of migrations. These models are based on the knowledge of the needs of customers per time slot. We have experimented our three methods on the BPM solution Bonita, and demonstrated that they provide substantial savings on the infrastructure exploitation compared to a basic method.

Keywords: elasticity, BPM, cloud, live migration