Allocation de ressources par une approche hybride apprentissage automatique et optimisation pour l'hébergement et l'infogérance de services web

Etienne Leclercq

UNIVERSITÉ PARIS XIII - SORBONNE PARIS NORD

École Doctorale Sciences, Technologies, Santé Galilée

Allocation de ressources par une approche hybride apprentissage automatique et optimisation pour l'hébergement et l'infogérance de services web

---

THÈSE DE DOCTORAT

présentée par

Etienne LECLERCQ

LIPN – Alter way / Smile

pour l'obtention du grade de

DOCTEUR EN INFORMATIQUE

soutenue le 17/10/2022 devant le jury d'examen composé de :

LEBBAH Mustapha, Professeur, Université Paris-Saclay ..... Président du jury
LOUDNI Samir, Professeur, IMT Atlantique ..... Rapporteur
MALICK Jérôme, Directeur de recherche, Université Grenoble Alpes ..... Rapporteur
ALES Zacharie, Enseignant-chercheur, ENSTA Paris ..... Examinateur
ROUVEIROL Céline, Professeure, Université Sorbonne Paris Nord ..... Directrice de thèse
ROUPIN Frédéric, Professeur, Université Sorbonne Paris Nord ..... Co-directeur de thèse
RIVALAN Jonathan, Responsable R&D, Smile ..... Encadrant industriel

Résumé

L'un des principaux défis du *cloud computing* est la gestion des ressources, qui s'inquiète principalement de la planification des charges de travail et du placement des services sur l'infrastructure. Ces dernières années, l'émergence de la virtualisation légère, à travers les conteneurs, a remis en avant le problème d'allocation des ressources, en introduisant notamment l'élasticité. Cette notion nouvelle pour la virtualisation a entraîné de nouvelles contraintes pour ce problème d'allocation, comme l'absence de connaissance a priori sur les ressources effectivement consommées par les individus.

Pour répondre à cette problématique et obtenir une gestion optimisée du placement de conteneurs sur une infrastructure, une méthodologie en deux étapes est développée, combinant des techniques d'apprentissage non supervisé et d'optimisation.

Une heuristique de placement *offline* est développée, utilisant un clustering des conteneurs en fonction de leur profil de consommation, afin d'obtenir une première solution de placement prenant en compte ces profils.

Cette solution de placement est mise à jour dans le temps par un système de boucle, combinant l'évolution du clustering avec des techniques d'optimisation dans un mode *online*. Le clustering est lui aussi maintenu à jour grâce à des méthodes d'optimisation en fonction de l'évolution de consommation des conteneurs.

Nous implémentons notre méthode sous la forme d'une librairie Python appelée Continuous Optimization for Time Series (COTS), dont l'interface d'entrée paramétrable permet son utilisation et évaluation dans différents contextes de jeux de données et de configurations.

À travers la définition et la comparaison de plusieurs critères liés au domaine de l'hébergement en micro-services,, nous évaluons l'efficacité de notre méthode par rapport à l'état de l'art.

Nous montrons ainsi une amélioration de l'efficacité énergétique allant de 5% à 30% en fonction des contextes métiers. Globalement, en considérant l'ensemble des critères, les performances sont améliorées en moyenne de $9,1\%$ par notre méthode. Une analyse détaillée de ces résultats et de divers indicateurs métiers ou algorithmiques précise le comportement de notre méthode et son utilisabilité sur différents scénarios hérités d'environnements de production réels.

Mots-clefs : optimisation, clustering, conteneurs, allocation de ressources, *cloud computing*

Abstract

One of the main challenge in *cloud computing* is resource management, mainly composed of scheduling workloads and services over the infrastructure. Recent years light virtualisation emergence, through containers, has brought back the resource allocation problem, notably by introducing elasticity. This new virtualization concept has led to new constraints for this allocation problem, such as the absence of a priori knowledge on the resources actually consumed by individuals.

To answer this problem and obtain an optimized management of containers assignment on an infrastructure, a two-step methodology is developed, combining unsupervised learning and optimization techniques.

An offline assignment heuristic is developed, using a containers clustering according to their consumption profile, in order to obtain a first assignment solution taking these profiles into account.

This assignment solution is updated over time by a *retro-action* loop system, combining the clustering evolution with optimization techniques in an online mode. Clustering is also kept up to date thanks to optimization techniques according to the containers consumptions evolution.

We implement our method through the Python library COTS, which configurable interface allows for its use and evaluation with various datasets and parameters.

Through the definition and comparison of several criteria related to the micro-services hosting domain, we provide the evaluation of our method efficiency compared to the state of the art.

Among these criteria, we show an improvement in energy efficiency from 5% up to 30%. Globally, considering all criteria, performance is improved by an average of 9.1% by our method. A deep analysis of these results and various domain or algorithmic indicators specifies of our method behavior and its usability on different use cases from a real production environment.

Keywords : optimization, clustering, containeurs, resources allocation, cloud computing

Remerciements

La réalisation de cette thèse est loin d'être le fruit de ma seule contribution et de mon seul travail. Il me semble donc nécessaire d'exprimer en premier lieu ma gratitude aux différentes personnes étant liées, de près ou de loin, à cette thèse.

Je tiens tout d'abord à remercier tout l'encadrement de cette thèse. Mes directeurs académiques, Céline Rouveirol, professeur de l'équipe A3 du LIPN, et Frédéric Roupin, professeur de l'équipe AOC du LIPN et directeur de l'Institut Galilée, qui m'ont fait confiance et m'ont soutenu, scientifiquement et moralement, pour permettre la réalisation de ces travaux.

Merci à Jonathan Rivalan, responsable R&D d'Alter way puis de Smile, qui m'a fait confiance pour réaliser cette thèse, m'a donné les moyens pour aller au bout de celle-ci et a su m'accompagner, tant sur le plan technique qu'individuel, quand il le fallait.

Je remercie les membres de mon comité de suivi qui ont veillé au déroulement de cette thèse. Merci à mon tuteur Antoine Rozenknop, à l'expert extérieur puis examinateur Zacharie Alès, au représentant de l'école doctorale Lionel Pournin, aux responsables du comité de suivi des thèses du LIPN Roberto Wolfler-Calvo puis Adeline Nazarenko. Je remercie également Samir Loudni et Jérôme Malick d'avoir accepté d'être rapporteurs pour cette thèse, ainsi que Mustapha Lebbah d'avoir accepté d'être le président de mon jury de thèse.

Je remercie l'ensemble de l'équipe R&D d'Alter way puis de Smile, pour toute l'aide et les échanges qui ont eu lieu au cours de cette thèse. Merci à Valentin Daviot et Damien Gilles pour avoir facilité mon intégration et pour leurs conseils techniques. Merci à Marco Mariani et Gilles Lenfant pour leurs conseils sur le développement de la librairie COTS. Merci à Veeraraghavan Sekar pour son aide sur le matériel, tout comme Saifuddin Mohammad. Merci à Sihem Cherrared, Hong Phuc Vu, Ulrich Koku Gblokpo pour tous les échanges au sein de l'équipe R&D.

Je remercie également l'ensemble des doctorants de l'école doctorale Galilée avec lesquels j'ai pu échanger au cours de cette thèse. Merci à Julie Sliwak pour les discussions sur nos situations, merci à Etienne Goffinet pour les échanges sur les points de concordance de nos sujets de thèse.

Merci à l'ensemble de mes amis, qui n'ont eu de cesse me motiver à aller au bout de ce projet. Les échanges avec eux m'ont permis de garder le sourire et l'envie pour continuer.

Je ne pourrais mentionner mes parents, mes sœurs, mon beau-frère et l'ensemble de ma famille. Ces personnes chères ont contribué, par leur soutien et tous les moments passés avec elles, à la réalisation et l'aboutissement de cette épreuve.

Enfin, je souhaitais finir en remerciant Clara qui a un réussi l'exploit de supporter mon

travail, ma situation et moi-même au quotidien durant toutes ces années. Elle a été le premier rempart aux moments difficiles, la première oreille à mon écoute, et la première voix à m'encourager, tout en étant la première à supporter les conséquences de ce défi.

Table des matières

Chapitre 1

Introduction

Sommaire

1.1 Contexte applicatif

1.1.1 Le cloud computing

Le *cloud computing* est devenu aujourd'hui un élément important de notre vie quotidienne, répondant à un grand nombre de besoins en informatique. Avec sa popularité grandissante, le terme *cloud computing*, ou souvent abrégé par *cloud* est largement utilisé pour décrire de nombreux aspects, liés par exemple à l'usage des infrastructures ou aux pratiques logicielles. Il est donc important de définir exactement à quoi fait référence le terme *cloud computing*, représentant le domaine d'application de cette thèse.

Une multitude de définitions existent pour le cloud [Vaquero et al., 2008, Geelan et al., 2009], ce qui s'explique principalement par le fait qu'il ne s'agit pas d'une nouvelle technologie, mais plutôt d'un nouveau modèle qui exploite un ensemble de technologies existantes pour répondre à de nouvelles exigences (économiques, applicatives...). La plupart des technologies utilisées par le cloud, notamment la virtualisation, ne sont pas nouvelles [Zhang et al., 2010]. Dans un effort d'uniformisation, l'Institut National des Standards et de la Technologie (National Institute of Standards and Technology) (NIST), un organisme international de standardisation, a adopté la définition suivante pour décrire le cloud computing (après traduction) [Mell et al., 2011] :

"Le cloud computing est un modèle permettant l'accès à un réseau à la demande, pratique et omniprésent, à un ensemble partagé de ressources informatiques configurables (e.g. des réseaux, des serveurs, du stockage, des applications ...) qui peuvent être provisionnées et libérées rapidement avec un effort de gestion minimal ou une interaction minimale avec les fournisseurs de services".

Cet organisme définit les caractéristiques principales du cloud comme suit [Mell et al., 2011] :

— Libre-service à la demande : un locataire peut utiliser unilatéralement des capacités informatiques en fonction de ses besoins, sans avoir nécessairement besoin d'une interaction humaine avec le fournisseur du service.

— Large accès au réseau : les services sont accessibles via le réseau par le biais de mécanismes standards qui encouragent l'utilisation de plateformes hétérogènes d'équipement légers ou lourds (téléphones mobiles, ordinateurs portables...).

— Mise en commun des ressources : les ressources informatiques du fournisseur sont mises en commun pour servir de multiples locataires. Le locataire n'a généralement aucun contrôle ou connaissance sur l'emplacement exact des ressources fournies, mais peut être en mesure de spécifier l'emplacement à un niveau d'abstraction plus élevé, par exemple le pays ou l'État.

— Élasticité rapide : les capacités peuvent être provisionnées et libérées automatique-

ment afin de s'adapter rapidement à la demande. Pour le locataire, les capacités disponibles semblent souvent illimitées.

— Service mesuré : les systèmes cloud contrôlent et optimisent automatiquement l'utilisation des ressources qui peut être surveillée et contrôlée, ce qui assure la transparence tant pour le fournisseur que pour le locataire.

Les services cloud sont souvent classés en fonction de la nature de ces services [Mell et al., 2011, Rani and Ranjan, 2014] :

— Infrastructure as a Service (IaaS) [Ambi Karthikeyan, 2018] : des services bas niveau, notamment des ressources de calcul et de stockage, sont fournis. Le fournisseur est responsable du réseau, des machines physiques et de la virtualisation, alors que le locataire est responsable du système d'exploitation et des applications.

— Platform as a Service (PaaS) [Pahl, 2015] : le locataire peut déployer des applications grâce à des outils de développement pris en charge par le fournisseur. Ce dernier est responsable du réseau, des machines physiques, de la virtualisation, des systèmes d'exploitation, des logiciels intermédiaires, des outils de développement et de la gestion de bases de données.

— Software as a Service (SaaS) [Stavrinides and Karatza, 2016] : le locataire a accès a des solutions applicatives complètes, gérées par le fournisseur à tous les niveaux.

En pratique, la distinction entre ces différents modèles peut être légèrement différente d'un fournisseur à un autre.

Ces services sont fournis en suivant trois types de déploiement différents, en fonction du type de cloud par rapport au client :

— Le cloud public [Ren et al., 2012] est une infrastructure destinée à une utilisation ouverte pour un grand public.

— Le cloud privé [Deshpande et al., 2014] est une infrastructure destinée à l'utilisation par une seule organisation. Par rapport au cloud public, il permet une meilleure sécurité, un meilleur contrôle et gestion des coûts, mais est moins performant lors d'un passage à l'échelle.

— Le cloud hybride [Srinivasan et al., 2015] est une composition de plusieurs infrastructures cloud (publiques et/ou privées). Le but du cloud hybride est de profiter des avantages des différents types de cloud selon les services considérés, ainsi que de renforcer les architectures applicatives en les redondant sur plusieurs infrastructures.

Le cloud désigne donc un domaine très large et grandissant, comme peut le montrer la présence des plus grandes entreprises sur ce marché : Google, Microsoft et Amazon font

partie des plus gros fournisseurs de services cloud. Si sa progression initale a été permise par des technologies de virtualisation dites de niveau 1 et 2, respectivement virtualisation depuis la machine physique (*bare metal*) et virtualisation depuis le système d'exploitation (Système d'exploitation (Operating System) (OS)), l'essor actuel du cloud repose sur un nouveau principe technologique dit de conteneurisation. Cette thèse s'inscrit dans ce cas précis rencontré dans le cloud : l'hébergement en conteneurs, dont la définition est détaillée par la suite.

1.1.2 Les services conteneurisés

Comme expiqué ci-dessus, la technologie de virtualisation est le centre des systèmes cloud, largement utilisés aujourd'hui pour répondre aux divers besoins informatiques, allant de la fourniture de simples capacités de calcul à des solutions complexes répondant à des besoins spécifiques.

La virtualisation est un processus d'exécution d'une instance virtuelle d'un système informatique dans une couche abstraite à partir d'un matériel physique (réel) [Inc., 2013]. Cela garantit la séparation des systèmes individuels pour renforcer leur sécurité et faciliter l'allocation des ressources. Il permet également d'émuler différents types de machine ou d'autres systèmes d'exploitation sur la machine physique existante avec le système d'exploitation existant.

Pendant longtemps, la virtualisation dans le monde de l'hébergement se voyait à travers la Machine Virtuelle (VM). Comme son nom l'indique, le principe d'une machine virtuelle est de virtualiser le fonctionnement d'une machine physique : chaque VM a son propre OS, ses propres ressources fixes, son ensemble de librairies et d'applications [Altintas et al., 2005]. Étant très utilisées ces deux dernières décennies, les différents challenges amenés par les VMs ont été très étudiés [Li et al., 2010], en particulier le problème d'allocation de ressources [Usmani and Singh, 2016].

Récemment, une autre forme de virtualisation a fait son apparition dans les solutions cloud : les conteneurs. Ce concept n'est pas nouveau, étant tiré des conteneurs Linux (ou LinuX Container (LXC) [Bernstein, 2014]), présents depuis longtemps sur le système d'exploitation éponyme. Un conteneur est une enveloppe virtuelle qui permet de *packager* une application avec toutes ses dépendances fonctionnelles : fichiers source, *runtime*, librairies, outils et fichiers. Ils sont packagés en un ensemble cohérent et prêt à être déployé sur un serveur et son système d'exploitation [Docker, 2013]. Contrairement à la virtualisation de serveurs et à une machine virtuelle, le conteneur n'intègre pas d'OS, il s'appuie directement sur le système d'exploitation du serveur sur lequel il est déployé, comme le montre la Figure 1.1 et décrit dans [Yadav et al., 2019]. Docker est un projet open source (et une entreprise aujourd'hui) proposant une sur-couche qui rend le développement et le déploiement des conteneurs beaucoup plus simple tout en

les standardisant [Docker, 2017a].

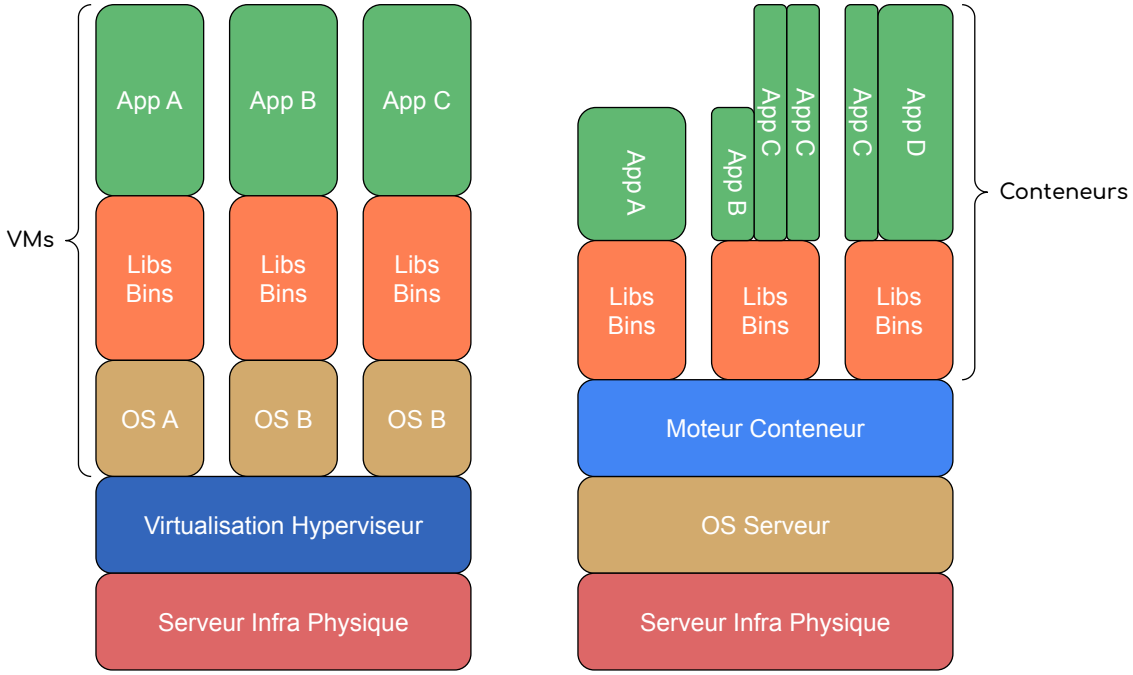


FIGURE 1.1 – Comparaison de la structure d'une VM et d'un conteneur.

Pour la gestion et le déploiement d'applications conteneurisées sur des serveurs Docker, un système d'orchestration de conteneurs comme *Kubernetes* [Kubernetes, 2018] est nécessaire. Initialement, Docker proposait son propre outil orchestration, appelé *Docker Swarm* [Docker, 2017b], avec des possibilités limitées pour la gestion des ensembles de conteneurs.

Le principal avantage des conteneurs est qu'ils introduisent moins de surcharge de virtualisation que les machines virtuelles, car il n'y a pas de couche de virtualisation supplémentaire. Les conteneurs sont donc considérés plus efficaces et permettent un meilleur passage à l'échelle. Cependant, cette différence dans les couches de virtualisation peut entraîner des risques liés à la sécurité [Mattetti et al., 2015, Young et al., 2019].

Les machines virtuelles et les conteneurs ont chacun leurs avantages et inconvénients, liés aux contraintes et exigences de leurs cas d'usages. Une combinaison de ces deux technologies de virtualisation est également possible, par exemple pour le déploiement d'un cluster de conteneurs au sein de machines virtuelles [Prakash et al., 2018, Sharma et al., 2016].

Par ailleurs, les conteneurs renforcent l'élasticité recherchée dans le cloud. L'élasticité est définie comme étant la capacité d'un système à ajouter et à supprimer des ressources, ou modifier leur configuration, pour s'adapter à la variation de charge en temps réel [Al-Dhuraibi et al., 2017]. Le terme élasticité est souvent associé à l'évolutivité, qui définit la capacité d'un système à supporter des charges de travail croissantes en utilisant

des ressources supplémentaires. Il est également associé à l'efficacité, qui dépend de la quantité de ressources consommées pour traiter une quantité de travail donné. L'allocation et le dimensionnement automatique visent à répondre aux compromis de la maximisation du bénéfice du fournisseur et la satisfaction des locataires. Un système élastique permet donc de satisfaire les variations de charge sur une infrastructure en limitant le gaspillage des ressources, c'est-à-dire le non-usage ou l'*overhead*, et permet de répondre à certains besoins ponctuels, comme l'*offloading*.

1.1.3 Contexte industriel

Alter way [Alter way, 2021], partenaire industriel de cette thèse, est un opérateur de services dédié au digital et aux systèmes d'information. Ses services incluent la création de sites internet, le conseil (design applicatif, technologie cloud, infrastructure), ou encore l'hébergement et l'infogérance de sites et applications web. L'entreprise est en pointe sur les plus récentes avancées technologiques : le cloud, *DevOps*, Open Data, l'industrialisation des plateformes PHP, l'accessibilité numérique et le web multi-plateformes. De plus, Alter way est largement impliqué dans le monde Open source et la promotion du Logiciel Libre, basant ses solutions sur des outils et une démarche open source et participant activement aux initiatives de l'écosystème. L'entreprise possède une équipe Recherche et Développement (RD), dont je fais partie, et qui a pour mission de renforcer l'outillage de la société ainsi que de développer des solutions innovantes. Elle poursuit une mission technologique simple : renforcer les systèmes existants (applications, services, plateformes) avec des composants *DevOps* facilitant leur amélioration par diverses méthodes issues de divers domaines (intelligence articielle, optimisation...).

Alter way migre actuellement depuis ses solutions d'hypervision Type 1 historiques, vers des systèmes en conteneurs, connus sous le nom de Caas (Container-as-a-service), ainsi que vers de nouvelles solutions de métrologie et d'alerting [Bhardwaj and Krishna, 2021]. Cette transformation permettra à Alter way d'élargir sa clientèle, tout en limitant le coût de la prise en charge.

Le Laboratoire d'Informatique Paris Nord (LIPN) et Alter way ont déjà collaboré dans le cadre du projet Wolphin [Alter way, 2017], fournissant un ensemble d'outils et mécanismes de métrologie, d'orchestration et de facturation, ainsi que les outils associés pour des infrastructures hébergeant des applications en conteneurs.

Le contexte métier a été introduit en définissant le cloud et la place des conteneurs dans celui-ci. La place grandissante de ces systèmes appporte de nombreuses questions et problématiques [Fatemi Moghaddam et al., 2015] comme la sécurité, la gestion et disponibilité des données, l'interopérabilité ou encore l'allocation de ressources [Selvi et al., 2014]. Cette dernière problématique constitue le cœur de cette thèse et est très

étudiée [Usmani and Singh, 2016, Tosatto et al., 2015] car elle met en jeu deux aspects opérationnels majeurs : la performance des applications et l'économie de ressources.

Aujourd'hui, ce dernier point connaît une forte résonance pour le secteur cloud, car il ouvre à la fois à une économie d'argent et à une économie d'énergie, constituant un levier d'optimisation dans le contexte écologique actuel. Fournir une solution permettant l'optimisation globale des ressources dans le contexte de la conteneurisation, semble donc être un objectif attendu à la fois par le domaine scientifique et industriel. Cet objectif et les problématiques associées sont détaillées par la suite.

Il est à noter que la société Alter Way a été rachetée par la société SMILE [SMILE, 2021] en Octobre 2021, et que la fin des travaux de cette thèse ont été réalisés sous la tutelle industrielle de cette dernière.

1.2 Contexte scientifique

Dans cette thèse, afin de répondre à nos problématiques (voir Section 1.3), nous proposons l'utilisation de méthodes tirées de deux domaines a priori différents : l'apprentissage et l'optimisation. L'apprentissage [Jordan and Mitchell, 2015] et l'optimisation [Bubeck, 2015] sont deux pièces essentielles des systèmes de prise de décision, appliqués à une grande variété de cas d'usages. Les méthodes de la première visent à obtenir des informations à partir de données ou d'expériences, celles de la seconde à obtenir la meilleure solution à un problème donné. Alors que chacun des deux domaines a beaucoup évolué, et chacun ayant sa propre communauté, des interactions sont apparues et les méthodes élaborées pour résoudre des problèmes impliquant ces deux domaines ont vu le jour [Song et al., 2019].

L'optimisation pour l'apprentissage

Premièrement, la plupart des problèmes rencontrés en apprentissage peuvent être modélisés par des problèmes d'optimisation, comme par exemple le clustering évoqué par [Ouali et al., 2016a] (voir en Section 2.1.2). Selon le but de la modélisation et le problème à résoudre, les algorithmes de machine learning peuvent être divisés en plusieurs catégories : apprentissage supervisé, apprentissage semi-supervisé, apprentissage non supervisé et apprentissage par renforcement. Plus précisément, l'apprentissage supervisé inclut les problèmes de classification (comme la classification d'images [Yang et al., 2009], ou de textes [Kim, 2014]) et de régression ; alors que l'apprentissage non supervisé contient notamment les problèmes de clustering et de réduction de dimension [Ding et al., 2002]. Pour chacune de ces familles, nous pouvons décrire l'objectif final comme la fonction objectif d'un problème d'optimisation. Dans [Sun et al., 2019], les auteurs décrivent les méthodes d'optimisation couramment utilisées pour les problèmes d'apprentissage

machine. On y retrouve l'utilisation d'algorithmes de gradient pour les Réseaux de neuronnes profonds (Deep Neural Network) (DNN) [J. Reddi et al., 2018, Keskar and Socher, 2017], mais aussi pour le méta-apprentissage [Andrychowicz et al., 2016], où l'optimisation peut être vue comme un problème d'apprentissage pour prédire le gradient plutôt que comme un algorithme de descente de gradient déterminé.

Dans [Bertsimas and Dunn, 2019], les auteurs montrent que des problèmes d'apprentissage (régression, classification) ont longtemps été résolus seulement par des heuristiques car ces problèmes n'étaient pas exactement attaquables pour des tailles et temps de calcul appropriés pour le domaine d'application. Ils montrent que les progrès dans l'optimisation, et notamment pour l'optimisation robuste ou l'optimisation en nombres entiers, permettent aujourd'hui d'envisager la résolution de ces problèmes de manière exacte.

L'apprentissage pour l'optimisation

Par ailleurs, l'apprentissage peut également être utilisé pour résoudre des problèmes d'optimisation [Bengio et al., 2021]. Dans ce cas, il est principalement utilisé pour accélérer le processus de recherche de solution et pour améliorer la qualité de cette solution. De manière générale, les techniques d'apprentissage sont utilisées pour extraire de la connaissance à partir des données recueillies lors de la résolution des processus d'optimisation, puis d'utiliser cette connaissance pour améliorer ou modifier une partie d'un algorithme d'optimisation [Zhang et al., 2011, Jourdan et al., 2006, Corne et al., 2012]. Par exemple, dans [Zhang et al., 2007] les auteurs utilisent un clustering pour trouver des sous-groupes au sein de la population d'un algorithme génétique, puis adaptent les autres paramètres en fonction de la taille des sous-groupes. Le choix ou l'ajustement dynamique des paramètres d'un algorithme d'optimisation constitue une part importante de l'utilisation des techniques d'apprentissage pour l'optimisation. Lors d'un algorithme de *branch-and-bound*, les méthodes d'apprentissage peuvent être utilisées pour aider le processus de deux étapes cruciales à cet algorithme : le choix des variables sur lesquelles brancher, et la sélection des nœuds de l'arbre généré par le *branch-and-bound* [Lodi and Zarpellon, 2017]. Dans [Morabit et al., 2021], les auteurs utilisent l'apprentissage pour accélérer cette fois une méthode de *génération de colonnes*. Dans celle-ci, l'apprentissage d'un modèle est appliqué à l'étape de sélection des variables (les colonnes) générées pour résoudre le problème maître réduit à chaque itération.

Parmi tous les travaux cités, nous remarquons que le lien entre les deux domaines est souvent matérialisé par une relation de service : les méthodes connues d'un domaine sont utilisées dans le but d'améliorer les méthodes existantes de l'autre.

1.3 Problématique

Le passage aux solutions en conteneurs amène de nouvelles contraintes inhérentes à cette technologie. L'élasticité propre aux conteneurs induit des consommations de ressources non régulières, et donc potentiellement des montées en charge mettant à contribution les infrastructures les accueillant, avec des risques de contention, voire de débordement. De plus, nous nous retrouvons à prendre en considération des entités (les conteneurs) très hétérogènes, sur différents aspects. En effet, l'effectif de candidats peut fortement varier (apparition et disparition), ainsi que leur durée de vie (certains services sont éxécutés en continu, d'autres sont éphémères - cas du Function as a Service (FaaS) [Fox et al., 2017]).

L'objectif de la thèse est donc l'obtention de méthodes permettant d'optimiser la gestion des conteneurs, en prenant en compte l'ensemble des contraintes métiers, permettant une économie en gestion et ressources utilisées, tout en limitant les incidents de production liés aux caractéristiques des conteneurs sans diminuer la qualité des services.

Le placement des conteneurs sur une infrastructure est une part importante du problème d'allocation de ressources en contexte de conteneurs. Ce problème se définit par l'assignation d'un ensemble de conteneurs $\{o_1, o_2, \ldots, o_I\}$, avec I étant le nombre de conteneurs présents sur l'infrastructure, à un ensemble de nœuds (ou serveurs) $\{n_1, n_2, \ldots, n_M\}$, avec M étant le nombre de nœuds disponibles sur l'infrastructure. Parmi les contraintes du problème de placement, nous avons :

- les contraintes de capacité : la consommation totale d'un nœud ne doit pas dépasser une limite (typiquement leur capacité maximale d'une ressource) ;
- les contraintes d'assignation : chacun des conteneurs doit être assigné à exactement un nœud (sauf en cas de redondance des applications, non considérée ici) ;
- le nombre de nœuds utilisés ne doit pas dépasser le nombre de nœuds disponibles sur l'infrastructure (ou une limite inférieure fixée)

L'objectif du placement peut refléter différents critères métiers : le nombre de nœuds utilisés, des taux de charge sur les nœuds utilisés, des critères de performance propres aux applications ...

Dans le domaine du cloud computing, la technologie des conteneurs s'est de plus en plus développée mais reste relativement nouvelle dans les systèmes de production. Le problème d'allocation de ces entités devient donc très étudié (voir Section 2.1.1) comme pour la technologie des machines virtuelles. Cependant, certaines problématiques ne sont pas ou peu considérées, la plupart des méthodes existantes dans ce contexte peuvant être classées en deux familles :

1. Les méthodes proposant une solution de placement au départ, sans adapter celle-ci en fonction de l'évolution de consommation des micro-services ;

2. Les méthodes proposant d'adapter les niveaux de consommation de ressources allouées aux conteneurs, mais pas le placement de ceux-ci sur l'infrastructure.

Alors comment proposer une solution de placement ajustable dans le temps en fonction de l'évolution des consommations de ressources par les conteneurs ?

Pour répondre à cette problématique, nous cherchons dans un premier temps à identifier des profils de consommation de ressources pour chaque conteneur. L'identification de ces profils vise à améliorer le placement des conteneurs grâce à deux facteurs :

— la stabilité du placement : en basant la technique de placement sur des profils de consommation, nous faisons l'hypothèse qu'un conteneur suivra un comportement similaire aux autres conteneurs de son profil. Une fois le placement établi à partir des profils, cette hypothèse nous permet d'espérer pas ou peu de déplacements de conteneurs (pour garantir la fiabilité du placement) ;

— la performance du placement : l'identification des conteneurs par un profil permet de représenter, dans le placement, un ensemble de conteneurs par un profil type, ce qui réduit la taille du problème considéré.

L'aspect *streaming* des données de consommations relevées dans notre cadre d'hébergement en conteneurs complexifie la problématique. En effet, une fois les profils de consommation identifiés, et la solution de placement appliquée, comment mettre à jour ces profils (et l'assignation des conteneurs à un profil type) ainsi que le placement courant ? C'est pour répondre à cette question que nous proposons une méthode alliant apprentissage non supervisé et optimisation. Les profils identifiés (grâce à l'apprentissage non supervisé) sont mis à jour grâce à des méthodes d'optimisation, et leur mise à jour entraîne à leur tour la mise à jour du placement, également via des méthodes d'optimisation. Ces méthodes d'optimisation incluent la modélisation des problèmes (de clustering et de placement) par des modèles d'optimisation et les notions de relaxation et de dualité.

L'élaboration de cette méthode répond donc à une double problématique métier : établir un placement prenant mieux en compte les différents types de consommation, et mettre à jour ce placement dans le temps en fonction de l'évolution des consommations. L'utilisation croisée de l'apprentissage et de l'optimisation constitue également un enjeu supplémentaire. En combinant les techniques issues de ces deux domaines, nous visons également à répondre aux contraintes opérationnelles du métier : en effet, se plaçant dans un contexte de données en *streaming*, les nouvelles solutions doivent être trouvées rapidement.

1.4 Plan de la thèse

Une représentation graphique du plan de ce manuscrit est visible en Figure 1.2.

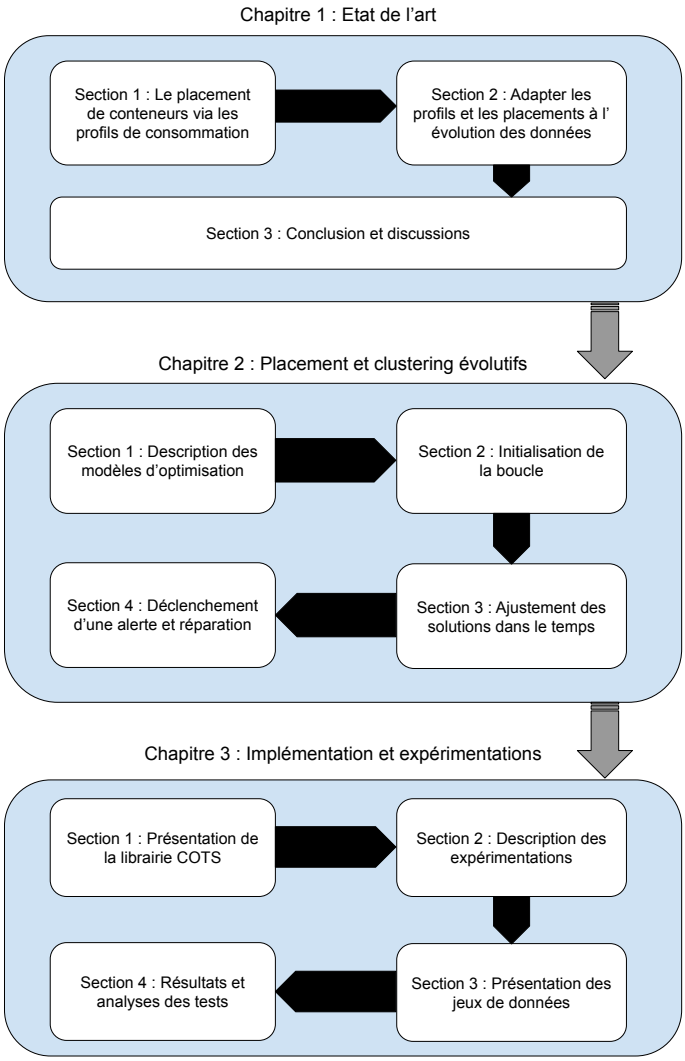


FIGURE 1.2 – Représentation graphique du plan

La première partie de ce manuscrit, le Chapitre 2, est dédiée à l'état de l'art réalisé au cours de la thèse. Dans celui-ci, nous commençons par discuter des méthodes existantes utilisées en pratique pour la gestion du placement de conteneurs dans une infrastructure. L'absence de suivi précis de l'évolution des consommations des conteneurs dans les méthodes de placement nous a permis de considérer un profilage des conteneurs en fonction de leur consommation. Ce profilage est possible grâce au clustering, problème d'apprentissage non supervisé dont les méthodes de résolution sont également étudiées. Dans un second temps, nous nous intéressons à l'aspect dynamique de notre problème : comment mettre à jour le placement, ainsi que les profils identifiés, en fonction de l'évolution des données de consommation. Cette problématique nous a amené à formuler nos problèmes de clustering et de placement par des problèmes d'optimisation, dont la littérature est étudiée. L'élaboration d'une méthode hybride alliant apprentissage

et optimisation constituant une contribution importante de la thèse, les interactions existantes dans la littérature entre ces deux domaines sont également discutées. Enfin, les concepts de base nécessaires à la compréhension de ce manuscrit sont introduits.

Dans un second temps, le Chapitre 3 détaille l'élaboration de notre méthode proposant une solution, mise à jour dans le temps, de placement de conteneurs sur une infrastructure. Cette méthode utilise un système de boucles alliant apprentissage non supervisé et optimisation pour ajuster dynamiquement une solution de placement. La modélisation des problèmes de clustering et de placement par des problèmes d'optimisation y est détaillée, ainsi que l'obtention, grâce à des heuristiques, de premières solutions à ces problèmes. Le fonctionnement de notre boucle, visant à mettre à jour ces solutions en fonction de l'évolution des données, est enfin détaillé.

Nous discutons ensuite en Chapitre 4 l'évaluation de notre méthode, à travers son implémentation via une librairie *Python* appelée COTS, et des expérimentations menées avec celle-ci. A travers ces expérimentations, nous analysons l'efficacité globale de la méthode grâce à l'identification de critères métiers, mais également son utilisabilité dans un environnement réel de production ou encore l'impact de l'évolution du clustering sur la solution de placement.

Les contributions de nos travaux sont enfin discutées en conclusion avec les perspectives ouvertes pour des travaux futurs. Cette contribution est illustrée à travers l'impact sur différents domaines :

— une méthode proposant une solution de placement dynamique pour les conteneurs répond à un besoin métier ;
— l'utilisation combinée de l'apprentissage et de l'optimisation renforce les interactions entre ces spécialités ;
— l'implémentation de cette méthode au sein d'une librairie visant à être publiée en open source constitue une contribution supplémentaire.

Chapitre 2

Etat de l'art

Sommaire

L'objectif de cette première partie est d'abord de présenter les différentes méthodes existantes pour la gestion du placement de micro-services dans une infrastructure cloud. Cet état de l'art, ainsi qu'un exposé des méthodes de l'état de l'art pour répondre à cette problématique du placement de services conteneurisés, nous ont amenés à identifier des profils de consommation et d'utiliser ces profils pour élaborer une stratégie de placement. Les méthodes permettant l'identification de ces profils, réalisée par le problème d'apprentissage non supervisé appelé clustering, sont également étudiées.

Dans un second temps, la dynamicité de nos problèmes (comment faire évoluer les profils et le placement associé quand les données évoluent) est étudiée, nous amenant à combiner des techniques d'apprentissage non supervisé et d'optimisation pour répondre à cette problématique. Les interactions entre ces deux domaines, et comment elles se matérialisent dans cette thèse, sont finalement discutées.

2.1 Le placement de conteneurs via les profils de consommation

2.1.1 L'allocation de ressources pour les conteneurs

Comme mentionné en Section 1.3, le problème d'allocation de ressources est très étudié dans le monde de l'hébergement et du cloud. La littérature concernant ce même problème pour les machines virtuelles est très fournie [Usmani and Singh, 2016], cependant la technologie des conteneurs apporte de nouvelles problématiques, comme l'élasticité [Al-Dhuraibi et al., 2017] qui entraîne des consommations potentiellement erratiques et non prévisibles.

Actuellement, le *scheduler* [Docker, 2016] (service dédié à l'attribution des conteneurs sur les différents nœuds de l'infrastructure) de Docker utilise des méthodes de placement relativement naïves :

— *random* : le placement des micro-services se fait de manière aléatoire sur les différents nœuds de l'infrastructure. Cette stratégie a l'avantage d'être très peu coûteuse, mais peut entraîner un placement déséquilibré des conteneurs sur l'infrastructure ;

— *spread* : le scheduler parcourt les conteneurs un par un, et les assigne sur les nœuds en les parcourant un par un également (le premier conteneur sur le premier nœud, le second conteneur sur le second nœud, etc...). Cette technique est utilisée par défaut et est également peu coûteuse. Elle apporte un équilibrage sur l'infrastructure en terme de nombre de conteneurs placés sur les différents nœuds, mais ne présente pas d'intelligence au vu des niveaux de ressources alloués aux conteneurs ;

— *binpack* : un rang est calculé à chaque nœud en fonction des ressources disponibles

(selon les demandes initiales des conteneurs présents) et le nombre de conteneurs présent sur ce nœud. Le conteneur est placé sur le nœud avec le meilleur rang.

Cependant, le scheduler de Docker Swarm ne prend pas en compte l'utilisation des ressources et son évolution au cours du temps lors du placement de conteneurs dans un cluster [Hassen, 2016].

Sur la base de l'équilibrage de charge, les auteurs de [Sureshkumar and Rajesh, 2017] ont proposé un algorithme de scheduling des conteneurs Docker pour contrôler dynamiquement la charge de chaque conteneur en considérant un certain seuil, afin que la charge de chaque conteneur soit bien équilibrée. Il considère les niveaux de ressources allouées, mais pas l'affectation des nœuds. Des *autoscalers* ont été pensés pour gérer les niveaux de ressources allouées aux conteneurs de manière automatique. De nombreux travaux ont été réalisés pour développer des *autoscalers* [Awada and Barker, 2017, Barna et al., 2017, Fokaefs et al., 2016], mais les autoscalers dépendent fortement des applications, et ne modifient pas non plus le placement des micro-services.

Concernant le placement des conteneurs sur l'infrastructure, plusieurs travaux ont également été réalisés [Casalicchio, 2016, Tosatto et al., 2015]. Dans [Kaewkasi and Chuenmuneewong, 2017], les auteurs ont proposé un algorithme d'ordonnancement de conteneurs basé sur un Algorithme de Colonies de Fourmis (Ant Colony Optimization) (ACO), dont le but est d'équilibrer l'utilisation de ressources afin que les applications dans les groupes de conteneurs aient de meilleures performances. Elle prend en compte les ressources réservées par conteneurs, et définit la solution d'affectation au début de la charge de travail.

Au cours de cette thèse, nous nous intéressons également à la modélisation de ce problème de placement de conteneurs par un problème d'optimisation. Parmi les travaux s'attaquant au placement des conteneurs, aucun ne le modélise par un modèle d'optimisation. Concernant les machines virtuelles, bien que d'autres aspects de l'allocation de ressources aient été plus étudiés (sélection de VMs pour migration, niveaux de ressources...), le placement de ces entités a été exploré, et il en existe des modélisations par des problèmes d'optimisation. Par exemple, dans [Rezvani et al., 2014], les auteurs proposent une formulation de ce problème via un programme linéaire en nombres entiers, pouvant servir de base pour modéliser notre problème (voir Section 3.2). Nous y retrouvons l'expression de contraintes définissant le problème de placement (contraintes d'assignation, de capacité ...) pouvant être réutilisées dans notre cas.

Un problème récurrent sur les solutions citées précédemment est l'ajustement dans le temps du placement des conteneurs sur les nœuds. En effet, l'élasticité des conteneurs peut entraîner des consommations irrégulières dans le temps et ainsi rendre le placement courant non adapté.

En prenant en compte toutes ces considérations, nous avons proposé une méthode dans

laquelle nous suivons l'évolution de consommation de ressources des conteneurs en les profilant. Les méthodes de l'état de l'art répondant au problème du placement de conteneurs sur une infrastructure n'utilisent pas le profilage de ces conteneurs.

L'identification de profils de consommation est possible grâce à des techniques d'apprentissage non supervisé regroupant des individus aux caractéristiques de consommation similaires, ces techniques sont dites de clustering. Les moyens d'obtenir un clustering sont discutés dans la partie suivante.

2.1.2 Le profilage d'individus : le clustering

Nous utilisons l'apprentissage pour identifier et regrouper des profils de consommation de ressources par les conteneurs. Ce regroupement d'individus en paquets homogènes, selon des attributs donnés, est possible grâce au partitionnement ou *clustering* [Bramer, 2007], un problème d'apprentissage non supervisé. À travers ce problème, nous cherchons à regrouper des objets similaires, tout en séparant les objets présentant des dissimilarités. Les groupes alors formés sont appelés *clusters*. Par cette définition, pour obtenir un bon clustering, il convient donc à la fois de :

— minimiser l'inertie intra-cluster pour obtenir des clusters les plus homogènes possibles ;

— maximiser l'inertie inter-cluster afin d'obtenir des clusters bien différenciés.

Cette notion de similarité (et dissimilarité) repose sur la définition d'une mesure de distance entre les objets. Il existe de nombreux moyens différents pour définir une distance entres deux objets X et Y [Pandit et al., 2011].

Parmi ceux-ci, la Distance Euclidienne (Euclidean Distance) (ED) [Liberti et al., 2014] est la distance la plus utilisée dans les travaux et articles publiés et se définit comme suit :

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.1}$$

La distance de Manhattan est aussi utilisée et est définie comme suit :

$$d = \sum_{i=1}^{n} |x_i - y_i| \tag{2.2}$$

Les distances Euclidienne et de Manhattan utilisent respectivement la norme d'ordre 2 et la norme d'ordre 1. Plus généralement, la distance de Minkowski utilise une norme d'ordre p et est définie comme suit :

$$d = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p} \tag{2.3}$$

D'autres distances [Golay et al., 1998] utilisent le coefficient de corrélation de Pearson, défini par :

$$cc = \frac{\sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})}{\sigma_x \sigma_y} \tag{2.4}$$

où $\overline{x}$ et σ_x sont respectivement la moyenne et l'écart type des points de l'objet x.

Un exemple très simple de clustering en deux dimensions est illustré en Figure 2.1.

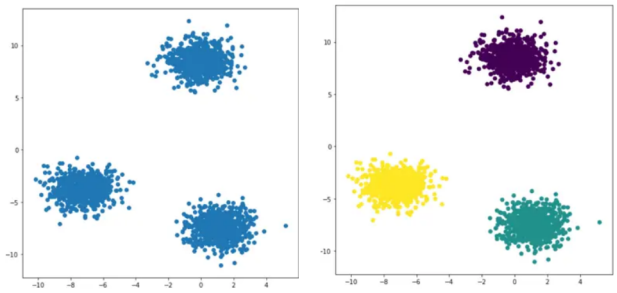


FIGURE 2.1 – Exemple d'un clustering de points de données, avec K=3

Les méthodes de clustering

Le clustering est un problème étudié depuis très longtemps, et qui peut être résolu par de nombreuses méthodes, notamment heuristiques [Xu and Wunsch, 2005, Xu and Tian, 2015], la complexité de ce problème rendant la résolution exacte difficile. Selon les critères utilisés, nous trouvons différentes classifications des algorithmes de clustering [Berkhin, 2006, Rai and Singh, 2010, Jain et al., 1999]. Deux principales familles d'algorithmes ressortent malgré tout : les méthodes de *partitionnement* et les méthodes *hiérarchiques*. Les premières nécessitent la définition de graines initiales pour former les clusters, qui sont ensuite améliorés de manière itérative. Les méthodes hiérarchiques, en revanche, peuvent utiliser les points de données individuels comme clusters uniques et construire le clustering. Le rôle de la mesure de distance est également différent dans ces deux algorithmes. Dans le clustering hiérarchique, la distance est initialement appliquée sur les points de données initiaux, puis progressivement appliquée sur les clusters en utilisant un point de données comme représentant de chaque cluster.

Cependant, dans le cas des méthodes de partitionnement, en général, les représentants des clusters choisis à différentes itérations peuvent être des points virtuels tels que le centroïde du cluster (n'inexistant pas dans les données initiales).

Nous détaillons par la suite le fonctionnement de ces deux familles d'algorithmes,

en spécifiant leurs caractéristiques, avant de nommer d'autres méthodes répandues n'appartenant pas à ces deux classes.

Méthodes de partitionnement

Les méthodes de clustering par partitionnement [Swarndeep Saket and Pandya, 2016] cherchent à regrouper les objets d'un jeu de données en optimisant une fonction objectif spécifique et en améliorant de manière itérative (et principalement de manière gloutonne) la qualité des partitions. Parmi ces méthodes, la plus répandue est l'algorithme *K-means* [MacQueen, 1967a,Lloyd, 1982]. Nous détaillons en premier lieu le fonctionnement de cet algorithme avant de présenter des variantes de ce dernier. L'algorithme *K-means* commence par choisir K points représentatifs comme centroïdes initiaux. Chaque point de donnée est ensuite affecté au centroïde le plus proche en fonction d'une mesure de proximité spécifique (e.g. distance Euclidienne comme décrit précédemment). Une fois les clusters formés, les centroïdes de chaque cluster sont mis à jour. L'algorithme répète ensuite itérativement ces deux étapes jusqu'à ce que les centroïdes ne changent pas ou que tout autre critère de convergence alternatif soit satisfait. Le clustering K-means est un algorithme glouton qui est garanti de converger vers un minimum local mais la minimisation de sa fonction de score est connue pour être un problème NP-difficile [Schütze et al., 2008]. En pratique, la règle d'arrêt au plus suivie est de poursuivre la procédure itérative jusqu'à ce que plus aucun des points de données (ou peu, typiquement 1% du nombre de points initiaux) ne changent leur appartenance à un cluster. Une preuve détaillée de la convergence mathématique des K-means peut être trouvée dans [Selim and Ismail, 1984].

Algorithme 1 : Algorithme *K-means*

Input : K nombre de clusters
Output : Clustering des données en K clusters

```
1 Sélectionner K points étant les centroïdes initiaux
2 while Critère de convergence non atteint do
3     Associer chaque point au groupe le plus proche, en considérant sa distance avec
        les centroïdes
4     Recalculer K nouveaux centroïdes c_k étant les centres des groupes c_k
5     c_k = 1/|c_k| ∑_{x∈c_k} x
```

Les étapes de l'algorithme K-means sont données en Algorithme 1, et une illustration de ce fonctionnement est visible en Figure 2.2. Pour mesurer la proximité d'un point aux différents centroïdes, plusieurs choix sont possibles (parmi les distances décrites précédemment), mais la distance Euclidienne est la plus utilisée.

La performance de l'algorithme K-means dépend fortement de deux facteurs :

— Le choix des centroïdes initiaux ;

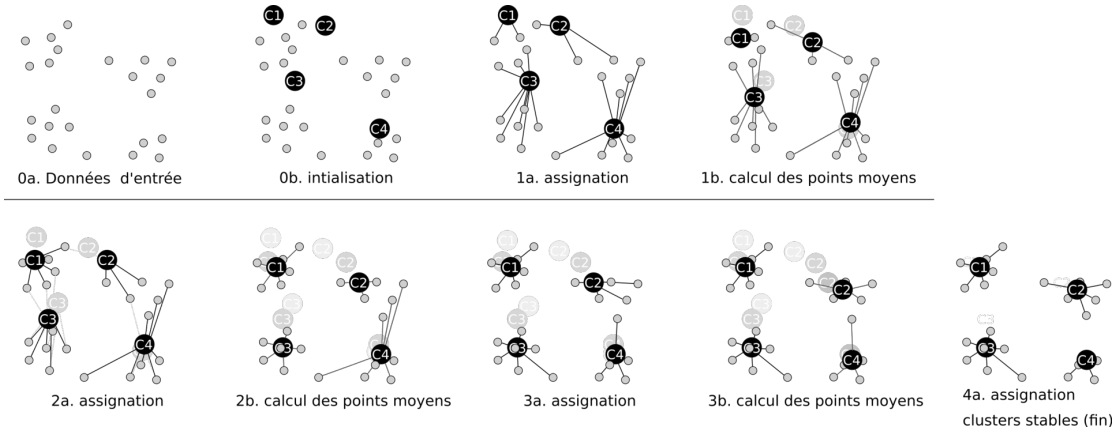


FIGURE 2.2 – Illustration du déroulement de l'algorithme K-means, avec K=4 (tiré de [Mquantin, 2017])

— Le nombre K de clusters.

De nombreux travaux proposent différentes manières de gérer ces deux points. En proposant initialement cet algorithme dans [MacQueen, 1967a], l'auteur utilise une solution simple pour le premier point : choisir les K graines initiales aléatoirement. Parmi les propositions d'initialisation, l'algorithme *K-means++* [Arthur and Vassilvitskii, 2006] initialise la première graine aléatoirement, puis sélectionne la graine suivante étant la plus éloignée de la première. La sélection des graines est poursuivie de cette manière jusqu'à obtenir les K centroïdes initiaux.

Le problème de choisir le "bon" nombre de clusters à utiliser est un des défis les plus importants pour le clustering en général. Plusieurs travaux proposent différentes méthodes pour y répondre, comme l'*Index de Calinski-Harabasz* [Caliński and Harabasz, 1974] ou encore le *Gap Statistic* [Tibshirani et al., 2001]. Cette problématique est détaillée en Section 4.2.

Beaucoup de méthodes de clustering se basent sur l'algorithme K-means en y apportant des modifications [Blömer et al., 2016, Yadav and Sharma, 2012]. Ces modifications portent généralement sur trois différents aspects de l'algorithme initial :

1. Choisir différents représentants des clusters (à la place des centroïdes) : la méthode *K-medoids* [Kaufman and Rousseeuw, 1990] par exemple utilise des points de données existants dans le jeu initial comme représentant des clusters et en calculant de manière itérative le coût d'échanger un représentant avec un point choisi aléatoirement ;
2. Choisir différentes méthodes d'initialisation pour les graines initiales [Celebi et al., 2013] ;
3. Appliquer des techniques de transformation des caractéristiques [Schölkopf et al., 1998].

L'algorithme K-means a comme premier avantage d'être simple dans son implémentation et dans son utilisation. Sa complexité est en moyenne en $O(knT)$, k étant le nombre de clusters, n le nombre d'individus et T le nombre d'itérations. Par ailleurs, son fonctionnement itératif est adapté au cadre dynamique de cette thèse. En revanche, pour utiliser ce type d'algorithme, l'utilisateur doit nécessairement fournir le nombre K de clusters à utiliser. De plus, cet algorithme n'est pas déterministe : si les centroïdes initiaux sont sélectionnés aléatoirement, le clustering final peut différer d'une exécution à une autre.

Méthodes hiérarchiques

Le *clustering hiérarchique* constitue la deuxième famille de méthodes de clustering les plus répandues [Murtagh and Contreras, 2011, Murtagh and Contreras, 2017, Nielsen, 2016]. Le qualificatif *hiérarchique* vient du fait qu'elle produit une hiérarchie H, l'ensemble des clusters à toutes les étapes de l'algorithme, en considérant l'ensemble des individus Ω qui vérifie les propriétés suivantes :

— $\Omega \in H$: au sommet de la hiérarchie, lorsque l'on groupe de manière à obtenir un seul cluster, tous les individus sont regroupés ;

— $\forall \omega \in \Omega, \omega \in H$: en bas de la hiérarchie, tous les individus se trouvent seuls ;

— $\forall (h, h') \in H^2, h \cap h' = \emptyset$ ou $h \subset h'$ ou $h' \subset h$: si l'on considère deux clusters du regroupement, alors soit elles n'ont pas d'individu en commun, soit l'une est incluse dans l'autre.

Cet aspect hiérarchique peut se représenter graphiquement par un dendrogramme. On le représente souvent comme un arbre binaire dont les feuilles sont les individus alignés sur l'axe des abscisses. Lorsque deux clusters ou deux individus se rejoignent avec l'indice d'agrégation τ , des traits verticaux sont dessinés de l'abscisse des deux classes jusqu'à l'ordonnée τ , puis ils sont reliés par un segment horizontal. À partir d'un indice d'agrégation τ , on peut tracer une droite d'ordonnée τ qui permet de voir une classification sur le dendrogramme. Un exemple de représentation de clustering hiérarchique par ce dendrogramme est visible en Figure 2.3.

Les méthodes de clustering hiérarchique se divisent en deux grandes familles : les méthodes *ascendantes* et les méthodes *descendantes*.

Le principe des méthodes hiérarchiques ascendantes est de démarrer d'une situation où tous les individus sont seuls dans un cluster, puis sont rassemblés en groupes de plus en plus grands. À chaque étape, deux clusters sont fusionnés, réduisant ainsi le nombre de clusters. Les deux clusters choisis pour être fusionnés sont ceux qui sont les plus *proches*, en d'autres termes, ceux dont la dissimilarité est minimale, cette valeur de dissimilarité étant appelée indice d'agrégation. Comme on rassemble d'abord les

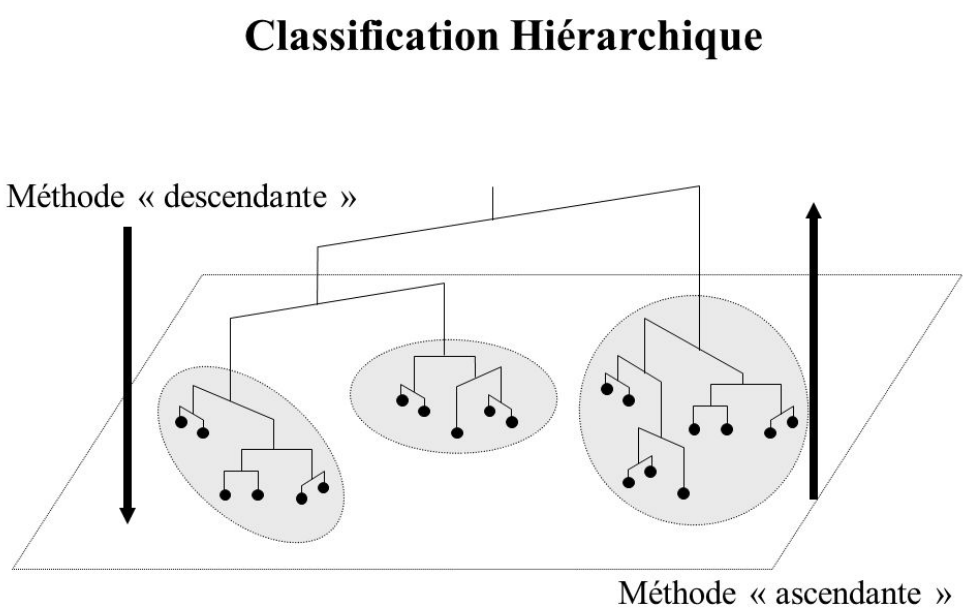
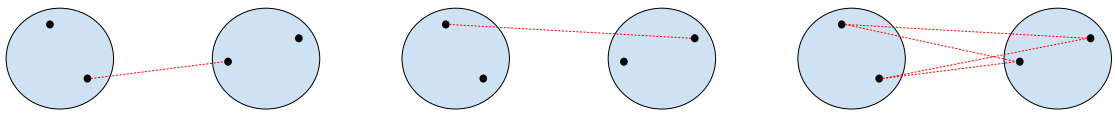


FIGURE 2.3 – Illustration d'un clustering hiérarchique

individus les plus proches, la première itération a un indice d'agrégation faible, mais celui-ci va croître d'itération en itération. Le clustering final est obtenu en fixant un indice d'agrégation maximal, ou en fixant le nombre de clusters à obtenir. Toutes les méthodes hiérarchiques ascendantes commencent par définir une matrice de dissimilarité, en utilisant une mesure de proximité définie. À chaque étape de fusion, cette matrice de dissimilarité est mise à jour en considérant la distance du cluster nouvellement formé avec les autres clusters.

Il existe plusieurs moyens pour définir la dissimilarité entre deux clusters [Jarman, 2020]. Les plus utilisés sont le *single-linkage clustering* [McQuitty, 1957] et le *complete-linkage clustering* [King, 1967]. Le premier nommé considère la distance entre les deux individus les plus proches appartenant à chaque cluster. Le second à l'inverse considère la distance maximale entre les paires d'individus appartenant aux deux clusters. En plus de ces deux mesures de dissimilarité entre clusters, nous trouvons également le *average-linkage clustering* [Sokal, 1958]. Ici la distance moyenne entre chaque paire d'individus appartenant à chacun des deux clusters est considérée.



(a) single-linkage (b) complete-linkage (c) average-linkage

FIGURE 2.4 – Exemple des différentes mesures de dissimilarité entre deux clusters.

La construction hiérarchique de ces méthodes permet de considérer une représentation globale des différents clusterings possibles des données. En effet, nous pouvons couper cette hiérarchie à n'importe quel niveau selon les critères (nombre de clusters souhaités, indice d'aggrégation maximal, etc...) et obtenir les clusters correspondants. Ceci différentie de manière significative le clustering hiérarchique du clustering par partitionnement, car le nombre de clusters n'est pas nécessairement fixé pour appliquer une telle méthode. Par ailleurs, contrairement au K-means, les méthodes hiérarchiques sont déterministes. Cependant, cette construction hiérarchique est plus coûteuse que l'algorithme *K-means*. De plus, alors que ce dernier améliore le clustering formé de manière itérative, le clustering hiérarchique présente moins de flexibilité : une fois que les fusions ou les divisions sont faites, elles ne peuvent pas être remises en cause. Ce défaut a par ailleurs été étudié dans [Fisher, 1996] en proposant une méthode itérative permettant la modification du dendrogramme créé jusqu'à l'obtention de la solution optimale. La complexité de ces méthodes hiérarchiques est quadratique, ce qui n'est pas souhaitable lorsque l'on considère de grands jeux de données.

Les méthodes basées sur la densité et les grilles

Des algorithmes de clustering basés sur la densité sont conçus pour découvrir des clusters de forme arbitraire [Kriegel et al., 2011]. Dans cette approche, un cluster est considéré comme une région dans laquelle la densité de points de données dépasse un certain seuil. DBSCAN et SSN sont deux des algorithmes les plus répandus de ce type de clustering.

L'algorithme DBSCAN a été introduit pour la première fois dans [Ester et al., 1996] et repose sur une notion de clusters basée sur la densité. Les clusters sont identifiés en regardant la densité de points. Les régions à forte densité de points reflètent l'existence de clusters tandis que les régions à faible densité de points indiquent des clusters de bruit ou des clusters de valeurs "aberrantes". Cet algorithme est particulièrement adapté pour traiter de grands ensembles de données, avec du bruit, et est capable d'identifier des clusters de tailles et de formes différentes. L'idée clé de l'algorithme DBSCAN est que, pour chaque point d'un cluster, le voisinage d'un rayon donné doit contenir au moins un nombre minimum de points, c'est-à-dire que la densité dans le voisinage doit dépasser un seuil prédéfini.

L'algorithme SNN [Ertöz et al., 2003], comme DBSCAN, est un algorithme de clustering basé sur la densité. La principale différence entre cet algorithme et DBSCAN est qu'il définit la similarité entre les points en examinant le nombre de voisins les plus proches partagés par deux points. En utilisant cette mesure de similarité dans l'algorithme SNN, la densité est définie comme la somme des similarités des plus proches voisins d'un point. Les points à haute densité deviennent des points centraux, tandis que les points à faible densité représentent du bruit. Tous les points restants qui sont fortement similaires à des points centraux spécifiques représenteront un nouveau ensemble de clusters.

Ces algorithmes basés sur la densité nécessitent la définition de plusieurs paramètres : la taille des listes de voisins considérés, un seuil de densité minimal, ou encore un seuil définissant les points centraux.

Les méthodes de clustering basées sur des grilles [Cheng et al., 2018] séparent l'espace des caractéristiques en un nombre fini de cellules qui forment une structure de grille sur laquelle toutes les opérations pour calculer le clustering sont effectuées. De manière générale, les méthodes de clustering par grille suivent les étapes présentées en Algorithme 2 [Grabusts and Borisov, 2002].

Algorithme 2 : Principales étapes d'un clustering par grille

1 Définir un ensemble de cellules
2 Assigner les objets aux cellules appropriées et calculer la densité de chaque cellule
3 Eliminer les cellules ayant une densité inférieure à un seuil fixé
4 Former des clusters à partir de cellules denses adjacentes

Les algorithmes les plus répandus de cette famille sont *CLIQUE* [Agrawal et al., 1998] et *STING* [Wang et al., 1997]. Cette approche est surtout utilisée pour des jeux de données multidimensionnels.

Les méthodes spectrales

Alors que les types de méthode décrites précédemment sont utilisées et étudiées depuis longtemps, le clustering spectral a gagné en popularité plus récemment, ayant comme origine les travaux décrits dans [Shi and Malik, 2000]. Les principales étapes d'un clustering sont données en Algorithme 3.

Algorithme 3 : Principales étapes d'un clustering spectral

1 Construire un graphe de similarité entre tous les points de données
2 Les données sont représentées dans l'espace engendré par certains vecteurs propres de la matrice de Laplace, en cherchant à minimiser un ratio où interviennent les coupes induites par la partition
3 Dans ce nouvel espace, un algorithme de clustering plus classique (e.g. K-means) est appliqué pour obtenir le clustering final

Les principes de base du clustering spectral sont détaillés dans [von Luxburg, 2007], où l'auteur part de la construction d'un graphe de similarité du problème traité, qui permet de construire une matrice de Laplace [Chung and Graham, 1997]. Il existe différentes variantes de clustering spectral [JingMao and YanXia, 2015], notamment dépendamment de la matrice de Laplace utilisée [Von Luxburg et al., 2008]. En considérant W une matrice de similarité d'un graphe G initial et D la matrice diagonale avec $D_i = \sum_j W_{ij}$, la matrice de Laplace normalisée L_N [Chung and Graham, 1997, von Luxburg, 2007] correspondante se définit comme suit :

$$L_N = I - D^{-1/2} W D^{-1/2} \tag{2.5}$$

La matrice de Laplace non normalisée L [Mohar et al., 1991] est, elle, définie par :

$$L = D - W \tag{2.6}$$

Les objectifs minimisés avec chacune de ces formes sont des ratios dans lesquels interviennent la coupe induite par la partition. Le ratio utilisé avec la matrice de Laplace normalisée L_N, appelé *normalized cut* et noté $Ncut$ est défini comme suit :

$$Ncut = \sum_{i=1}^{K} \frac{s(\Gamma_i, \Gamma_i^c)}{\sum_{j \in \Gamma_i} d_j} \tag{2.7}$$

Le ratio utilisé avec la matrice de Laplace non normalisée L, appelé *ratio cut* et noté $Rcut$, est défini par :

$$Rcut = \sum_{i=1}^{K} \frac{s(\Gamma_i, \Gamma_i^c)}{|\Gamma_i|} \tag{2.8}$$

avec K le nombre de clusters, Γ_i le $i^{\text{ème}}$ cluster, Γ_i^c le complémentaire de Γ_i dans G, $s(A, B) = \sum_{i \in A} \sum_{j \in B} W_{ij}$ et $d_i = \sum_j W_{ij}$.

Souhaitant une partition en K groupes, l'extraction des K premiers vecteurs propres (correspondant aux K plus petites valeurs propres) de cette dernière matrice permet la considération d'un espace en K dimensions. Finalement, le partitionnement est réalisé en considérant les lignes de cette matrice réduite comme représentant des objets, et regroupant ces derniers en K groupes (via l'algorithme de K-means par exemple).

Le clustering spectral, par les étapes supplémentaires utilisant la matrice de Laplace associée à un graphe, est plus coûteuse. Par ailleurs, en utilisant les ratios décrits ci-dessus, l'approche spectrale forme des clusters plus équilibrés en terme de taille.

Le clustering de séries temporelles

Les données que nous utilisons dans cette thèse sont des courbes d'évolution de consommations de ressources, et sont représentées par des séries temporelles. Une série temporelle est décrite comme étant une séquence de nombres réels représentant les mesures d'une variable à intervalles de temps réguliers [Gunopulos and Das, 2000]. On retrouve des séries temporelles dans de nombreuses applications, comme la météo avec l'évolution de températures et autres indicateurs, les finances avec les variations des cours boursiers, etc ... C'est pourquoi le cas particulier du clustering de séries temporelles a également été très étudié [Warren Liao, 2005, Rani and Sikka, 2012].

Le clustering de séries temporelles est un défi important à cause de la difficulté à définir une distance ou une similarité entre différentes séries temporelles pouvant présenter des caractéristiques différentes, à la fois sur l'aspect temporel et sur l'aspect comportemental (forme de la série). Les distances usuelles (e.g. distance Euclidienne) telles que décrites précédemment restent très utilisées pour ce cas particulier. Cependant, de nouvelles distances peuvent être utilisées [Ding et al., 2008, Agrawal et al., 1993].

Ces études classifient les mesures de distance selon ce qui est comparée :

— les mesures comparant directement les caractéristiques (*feature-based*) [Warren Liao, 2007, Kumar et al., 2002, Vlachos et al., 2003] : dans ce cas les séries temporelles sont considérées comme des vecteurs, chaque point de donnée étant une caractéristique. Les méthodes de clustering sont donc appliquées sur ces vecteurs. Les distances usuelles sont souvent utilisées dans ce cas ;

— les mesures comparant les formes des séries temporelles (*shape-based*) [Meesrikamolkul et al., 2012] : la comparaison ne se fait pas point à point, mais recherche des similitudes dans les formes, par un étirement et une contraction linéaires des axes temporels. Les algorithmes de cette famille utilisent généralement des méthodes de clustering classiques, mais en utilisant une mesure de similarité adaptée. Les distances les plus connues sont DTW et $LCSS$, qui sont décrites ci-dessous ;

— les mesures utilisant des modèles construits à partir des données brutes (*model-based*) [Xiong and Yeung, 2002, Wang et al., 2002, Baragona, 2001] : ici les séries temporelles sont transformées en modèles (un modèle paramétrique par série), sur lesquels on effectue le clustering en utilisant une distance entre eux [Kalpakis et al., 2001] ainsi qu'une méthode (souvent choisie parmi les méthodes classiques) [Warren Liao, 2005]. Cependant, il est montré que ces approches ont généralement des problèmes de scalabilité [Vlachos et al., 2004], et leurs performances diminuent lorsque les clusters sont proches les uns des autres [Mitsa, 2010].

La mesure de similarité la plus utilisée pour identifier les similarités de forme des séries temporelles est la Déformation dynamique temporelle (Dynamic Time Warping) (DTW) [Senin, 2008]. Pour comparer deux séries temporelles, au lieu de comparer les séries point par point à chaque instant t, DTW permet la comparaison d'un point d'une série à un ou plusieurs points de l'autre série. Ceci permet l'identification de décalage temporel de données similaires d'une série à une autre. En considérant deux séries x et y de longueurs respectives T_x et T_y, la distance $D(i, j)$ entre deux sous-séquences $x_1, x_2, \ldots, x_i$ et $y_1, y_2, \ldots, y_j$ se définit de manière récursive telle que :

$$D(i,j) = \begin{cases} |x_1 - x_1| & \text{si } i = j = 1 \\ |x_i - x_j| - \min\{D(i-1,j), D(i-1,j-1), D(i,j-1)\} & \text{sinon} \end{cases}$$

La mesure de similarité $DTW(x, y)$ entre les séries x et y se définit finalement comme suit :

$$DTW(x, y) = \frac{1}{D(T_x, T_y)} \tag{2.9}$$

Cette mesure est très utilisée pour regrouper des séries ayant des motifs similaires, sans prendre en compte la notion de temps (deux séries présentant des formes similaires mais décalées dans le temps auront tendance à être regroupées dans le même cluster). A l'inverse, les distances "classiques" (e.g. distance Euclidienne) regrouperont des séries présentant des similitudes dans le temps.

Les méthodes de clustering de séries temporelles suivent généralement la même classification que pour les méthodes de clustering classique : les méthodes par partitionnement, les méthodes hiérarchiques ou encore les méthodes basées sur la densité [Aghabozorgi et al., 2015].

Parmi les méthodes par partitionnement, la méthode phare K-means [MacQueen, 1967b] telle que décrite en Section 2.1.2 reste très utilisée pour les séries temporelles. Elle est souvent utilisée avec la distance Euclidienne comme mesure de similarité, mais des travaux se sont intéressés à son utilisation avec le DTW comme mesure de proximité [Niennattrakul and Ratanamahatana, 2007]. Cependant, le calcul de cette mesure étant coûteux, l'algorithme perd l'avantage d'être peu coûteux en utilisant cette dernière. Par ailleurs, les variantes de l'algorithme K-means sont également utilisées pour les séries temporelles, comme *K-medoids (PAM)* [Kaufman and Rousseeuw, 1990] qui utilise les données d'une série comme représentant des clusters au lieu de la moyenne des séries du cluster.

De la même manière, les méthodes hiérarchiques telles que décrites en Section 5 sont également directement utilisées pour le clustering de séries temporelles. Elles offrent un pouvoir de visualisation important [Keogh and Pazzani, 1998], et l'utilisation de distances alternatives (e.g. DTW) est facilitée de telles méthodes. Cependant, la complexité de tels algorithmes rend leur utilisation difficile pour des jeux de données importants. Dans [Bradley et al., 2002], les auteurs montrent que les algorithmes K-means ou K-medoids sont très rapides par rapport aux méthodes hiérarchiques. Une utilisation combinée de ces deux types de méthodes a été étudiée dans [Hautamaki et al., 2008], où les auteurs proposent une méthode hiérarchique suivie d'un K-means adapté utilisant la distance DTW. Les représentants de clusters utilisés sont les distances DTW moyennes des objets appartenant à ce cluster, dont l'imprécision a été montrée dans [Niennattrakul and Ratanamahatana, 2007].

2.1.3 Discussion

Après une revue de la littérature concernant les méthodes de placement pour les conteneurs, aucune d'entre elles n'utilise l'évolution des consommations de ressources pour élaborer une stratégie de placement. Cette évolution dans les caractéristiques des conteneurs n'est prise en compte que pour des méthodes ajustant le niveau de ressources alloués à ceux-ci. Notre proposition de stratégie de placement utilisant les profils de consommation des conteneurs est donc originale de ce point de vue.

Nous avons donc étudié les méthodes de clustering, notamment de séries temporelles, dans le but d'identifier des profils de consommation.

Les méthodes de clustering décrites dans un premier temps dans un cas général (notamment des données statiques) sont classées par leur type d'algorithme. Les méthodes de partitionnement, représentées par l'algorithme K-means, présentent une complexité moindre comparée à la plupart des autres méthodes.

Dans [Gulati et al., 2015], les auteurs comparent différentes méthodes de clustering, notamment *K-means* et des méthodes hiérarchiques. Il est montré que les méthodes hiérarchiques sont très coûteuses lorsque les jeux de données utilisés sont importants, à cause de la formation des dendrogrammes permettant la formation des clusters. Or dans notre cas, le nombre de conteneurs à gérer peut être très grand (plusieurs centaines de milliers) tout comme le nombre de points de données considérés, dépendant des fenêtres de temps observées.

Cette complexité se retrouve également dans les méthodes spectrales qui utilisent des transformations des données initiales (matrice de Laplace) puis calcule les vecteurs propres de cette matrice. Ces étapes supplémentaires peuvent engendrer des temps de calcul supplémentaires par rapport au *K-means*, qui constitue lui même la dernière étape du clustering spectral. De plus, l'une des caractéristiques du clustering spectral est de favoriser l'équilibre des clusters formés en terme de taille. Or, cet équilibrage n'est pas nécessaire dans notre problématique.

Parmi les inconvénients de K-means, on notera la nécessité de fournir un nombre de clusters à utiliser. Plusieurs travaux essaient de répondre à cette problématique. Certains travaux cherchent également à réduire l'aspect aléatoire de la méthode K-means, l'algorithme n'étant pas déterministe, notamment en modifiant l'initialisation des graines. Malgré ces limitations, l'algorithme *K-means* reste efficace et très attractif.

Alors que la distance Euclidienne ou les normes de type L_p effectuent une comparaison point par point entre les séries, d'autres mesures comme le DTW ont été introduites permettant une comparaison élastique des séries. Des similarités dans la forme des séries, même à des temporalités différentes, sont mises en valeur dans ce cas. Les distances usuelles comportent certains avantages de par leur simplicité, leur temps de calcul

linéaire et le fait qu'elles ne nécessitent aucun paramétrage. Dans notre cas, nous souhaitons privilégier l'aspect temporel des séries (la concordance de pics ou creux de consommation au même moment), c'est pourquoi les distances usuelles, comme la distance euclidienne, seront préférées.

Nous proposons une méthode originale de placement utilisant le résultat de clustering à la Section 3.3. Le cadre de cette thèse, impliquant l'arrivée continue de nouvelles données de consommation, nous oblige à remettre en cause de manière synchrone ou asynchrone (alerte) le placement et/ou les profils identifiés. Nous pensons que l'algorithme *K-means*, par sa méthode itérative et sa rapidité d'exécution, est alors particulièrement adapté dans ce cadre. Cette dynamicité est étudiée dans la section suivante.

2.2 Adapter les profils et les placements à l'évolution des données

2.2.1 La dynamicité du clustering et du placement

Nous avons deux types de problèmes dynamiques à traiter dans notre méthode : le clustering et le placement de conteneurs.

Comme détaillé en Section 2.1.1, les seuls travaux proposant des solutions de placement de conteneurs ne permettent pas d'ajuster dynamiquement ce placement. La méthode de placement dynamique de conteneurs est donc novateur pour le métier, dont les replacements nécessaires de conteneurs (incidents de production, etc...) sont réalisés directement par l'expert (couramment un administrateur système). C'est pour cette raison que nous utilisons le profilage des conteneurs pour établir notre stratégie de placement : en suivant l'évolution des consommations par ce profilage, nous faisons l'hypothèse que les déplacements nécessaires dans le placement pourront être anticipés ou effectués de manière réactive. Mais les profils des conteneurs doivent également être adaptés suivant l'évolution des consommations dans le temps.

Le clustering de données évolutives est un problème largement étudié ces dernières années. En effet, nous trouvons dans la littérature des études générales sur le clustering de données en streaming [Aggarwal, 2018, Nguyen et al., 2015, Yogita and Toshniwal, 2013, Mousavi et al., 2015]. Les principaux défis d'un tel clustering sont décrits dans [Khalilian, 2010], alors qu'une taxonomie pour classer ces algorithmes de clustering de données en streaming a été introduite dans [Silva et al., 2014].

La notion de données dynamiques, évoluant au fil du temps dans un processus de *streaming* apporte de nouveaux défis, comme le volume souvent important de données arrivant ou l'aspect évolutif des données [Aggarwal, 2005]. Ces challenges empêchent l'utilisation directe de la plupart des méthodes classiques de clustering. C'est pourquoi

dans [Aggarwal et al., 2003b], les auteurs proposent une méthode divisée en deux phases : une phase *online* qui stocke périodiquement une abstraction des données, et une phase *offline* qui applique une méthode de clustering sur cette abstraction. Les auteurs définissent cette abstraction des données par deux structures principales :

— *Microclusters* : des informations statistiques des données sont stockées par des *microclusters*, définis comme une extension du *cluster feature vector* [Zhang et al., 1996] ;

— *Pyramidal Time Frame* : les microclusters sont stockés par *snapshots* dans le temps suivant une structure pyramidale.

Beaucoup de méthodes utilisent cette séparation en deux phases en modifiant la technique de clustering utilisée : par exemple, *DenStream* [Cao et al., 2006] utilise l'algorithme *DBSCAN*, ou encore *StreamKM++* [Ackermann et al., 2012] utilise le *K-means* (ou plutôt le *K-means++* [Arthur and Vassilvitskii, 2006], une alternative au *K-means* classique). Dans [Ghesmoune et al., 2016], les auteurs catégorisent les algorithmes en fonction de la nature de la méthode sous-jacente utilisée pour le clustering. Parmi ces différentes classes de clustering, nous retrouvons sans surprise les mêmes familles de méthodes que celles pour le clustering classique décrites en Section 2.1.2 :

— Les méthodes basées sur le clustering par partitionnement : la méthode de clustering par partitionnement la plus utilisée étant le *K-means*, nous retrouvons beaucoup de méthodes adaptant cette dernière aux données dynamiques [Aggarwal et al., 2003a, Ackermann et al., 2012, Ailon et al., 2009] ;

— Les méthodes basées sur le regroupement hiérarchique [Kranen et al., 2011, Udommanetanakit et al., 2007, Meesuksabai et al., 2011] ;

— Les méthodes basées sur la densité [Amini et al., 2014] ([Cao et al., 2006]) ;

— Les méthodes basées sur le clustering en grille [Chen and Tu, 2007, Wan et al., 2009] ;

Une étude comparative de la plupart de ces méthodes a été réalisée dans [Kokate et al., 2018]. Les auteurs y montrent qu'aucune méthode ne domine les autres, et que les performances et la qualité du clustering calculé est très dépendant des caractéristiques de l'application.

Afin de permettre l'ajustement des profils ainsi que du placement associé, nous proposons un système de boucle utilisant des techniques d'apprentissage non supervisé et d'optimisation. L'utilisation combinée de techniques issues de ces deux domaines est discutée par la suite.

2.2.2 L'utilisation combinée de l'apprentissage non supervisé et de l'optimisation

Alors que l'émergence de liens entre l'apprentissage et l'optimisation a été discutée en Section 1.2, nous nous intéressons ici plus précisément aux liens existant entre l'apprentissage non supervisé [Cornuéjols and Miclet, 2011], représenté par le clustering permettant l'identification de profils, et l'optimisation, représenté par la modélisation de problèmes mathématiques et l'utilisation de propriétés spécifiques.

Problème d'optimisation, relaxation continue et dualité

Dans un premier temps, nous introduisons dans cette partie des notions et propriétés fondamentales en optimisation et nécessaires à la bonne compréhension de la suite de ce chapitre.

Nous décrivons un problème d'optimisation de la forme :

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g(x) \leq 0 \\ & x \in \Omega \end{array} \tag{2.10}$$

Ce problème consiste à trouver un élément $x^* \in \Omega$ qui minimise la fonction f tout en respectant les contraintes $g(x^*) \leq 0$. La valeur $f(x^*)$ est alors la valeur optimale du problème. La fonction f est appelée la fonction objectif. L'ensemble $\{x \in \Omega : g(x) \leq 0\}$ est l'ensemble des solutions réalisables. Si cet ensemble est vide, alors le problème est non réalisable.

En recherche opérationnelle, la relaxation continue d'un Programme Linéaire en Nombres Entiers (PLNE) consiste à relâcher les contraintes d'intégrité de ce programme. Par exemple, dans un problème d'optimisation en nombres entiers, les contraintes peuvent être de la forme :

$$x_i \in \{0, 1\} \tag{2.11}$$

La relaxation continue d'un tel problème consiste à remplacer ces contraintes par les suivantes :

$$0 \leq x_i \leq 1 \tag{2.12}$$

Cette relaxation est souvent utilisée dans le but d'obtenir une borne inférieure (dans le cas d'une minimisation) de la solution optimale du problème initial discret. En

effet, la minimisation continue se fait sur un ensemble contenant l'ensemble des points admissibles du problème discret. Le domaine admissible d'un problème d'optimisation en nombres entiers et de sa relaxation continue est illustré en Figure 2.5. Lors d'une optimisation linéaire en nombres entiers, la relaxation continue peut s'avérer à la fois efficace et facile à mettre en œuvre.

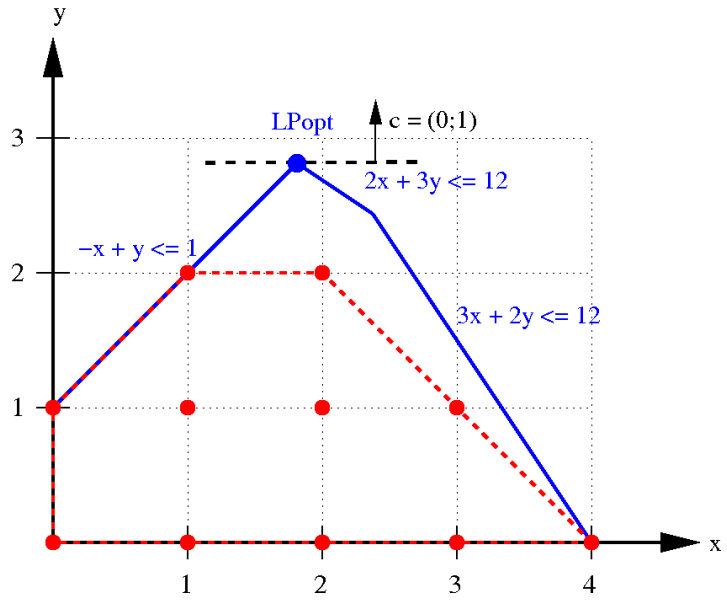


FIGURE 2.5 – Domaine admissible discret (en rouge) et sa relaxation continue (en bleue), tiré de [Sdo, 2018].

En optimisation continue, considérer le problème dual d'un problème [Tind and Wolsey, 1981, Geoffrion, 1974] est une approche très courante qui permet souvent d'élaborer des algorithmes de résolution, ou pour le moins d'obtenir des bornes pour le problème initial.

En optimisation linéaire continue, la fonction objectif du problème primal est une combinaison linéaire de n variables, soumis à un ensemble de m contraintes. Dans le problème dual, la fonction objectif devient une combinaison linéaire de m variables, soumis à un ensemble de n contraintes. Un problème primal générique et son problème dual est illustré en Figure 2.6.

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i & & \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{j,i} x_i \geq b_j & \forall j \in [1;m] \\ & x_i \geq 0 & \forall i \in [1;n] \end{aligned}$$

(a) Problème primal

$$\begin{aligned} \max \quad & \sum_{j=1}^{m} b_j y_j & & \\ \text{s.t.} \quad & \sum_{j=1}^{m} a_{j,i} y_j \leq c_i & \forall i \in [1;n] \\ & y_j \geq 0 & \forall j \in [1;m] \end{aligned}$$

(b) Problème dual

FIGURE 2.6 – Comparaison d'un problème primal et son dual.

Concrètement, pour chaque contrainte du problème primal, la valeur de sa variable duale associée dans le problème dual est le coût marginal associée à cette contrainte dans le problème primal. Une variable duale élevée signifie donc que la contrainte à laquelle elle est associée est *active* et donc qu'en supprimant cette contrainte dans le problème primal, nous obtenons la solution d'un problème qui est une relaxation du problème initial.

Modélisation du clustering via l'optimisation

Nous avons vu en Section 2.1.2 les méthodes heuristiques utilisées pour calculer un clustering. Ce problème d'apprentissage non supervisé a déjà fait l'objet d'une modélisation par un problème d'optimisation. Afin de faciliter la compréhension des formulations possibles de nos problèmes de clustering et de placement par des PLNE, nous rappelons dans le Tableau 2.1 les notations utilisées pour décrire ces problèmes.

Notation	Signification
I	Nombre d'individus (conteneurs)
K	Nombre de clusters (clustering)
$\{o_1, o_2, \ldots, o_I\}$	Ensemble des individus
$\{c_1, c_2, \ldots, c_K\}$	Ensemble des clusters
$w_{i,j}$	Distance entre les individus o_i et o_j

Tableau 2.1 – Index des notations pour les modèles d'optimisation

.

Le clustering sous contraintes [Basu et al., 2008] est le type de clustering le plus modélisé par un PLNE [Mueller and Kramer, 2010]. Tout comme la programmation par contraintes [Rossi et al., 2008], qui est largement utilisée pour modéliser et résoudre cette famille de clustering [Duong et al., 2017], le PLNE facilite la description des contraintes. Par exemple, dans [Tang et al., 2020], les auteurs s'intéressent au clustering à tailles de clusters fixes, tailles facilement représentées par des contraintes dans le modèle. Dans [Babaki et al., 2014], le clustering sous contraintes est résolu par une méthode de génération de colonne, dans laquelle un problème maître permet le choix de clusters parmi un ensemble de candidats générés par un problème esclave. Cette méthode est également utilisée pour le *minimum sum-of-squares clustering* (MSSE) dans [Aloise et al., 2012]. Le choix de clusters parmi un ensemble de clusters potentiels est également l'objectif dans [Ouali et al., 2016b], où les auteurs s'intéressent au clustering conceptuel, qui construit en parallèle des groupes et leurs caractérisations sous forme de conjonction d'attributs booléens. Dans [Dao et al., 2018], les auteurs modélisent le clustering descriptif comme un problème d'optimisation bi-objectif, et la comparent avec une formulation via programmation par contraintes qui s'avère plus efficace. Dans ces formulations, la variable de décision importante sert à affecter les individus à un cluster et est de taille $I \times K$. Cette variable est du type :

$$y_{i,k} = \begin{cases} 1 & \text{si le conteneur } o_i \text{ appartient au cluster } c_k \\ 0 & \text{sinon} \end{cases}$$

Cependant, dépendamment du contexte et de l'application, il n'est pas toujours possible de considérer un ensemble de clusters candidats. C'est pourquoi la plupart des heuristiques couramment utilisées pour résoudre un clustering sont *assemblistes* : les individus sont regroupés en fonction de leur similarité (ou dissimilarité), sans être assignés à un cluster précis. Certaines des heuristiques les plus connues ont fait l'objet d'une formalisation via l'optimisation linéaire : K-means [Ágoston and E-Nagy, 2021], méthodes hiérarchiques [Gilpin et al., 2013, Gilpin and Davidson, 2017]. Dans ce cas, la variable de décision primordiale à utiliser est de taille $I \times I$ et est du type :

$$u_{i,j} = \begin{cases} 1 & \text{si } o_i \text{ et } o_j \text{ sont dans le même cluster} \\ 0 & \text{sinon} \end{cases}$$

Comme expliqué en Section 2.1.2, un clustering a classiquement deux objectifs : minimiser la somme des distances intra-cluster (*Intra-Cluster Similarity, ICS*) et maximiser les distances inter-clusters (*Inter-Clusters Dissimilarity, ICD*). La fonction objectif des PLNE modélisant un problème de clustering utilise donc souvent l'un des deux objectifs ci-dessus, respectivement définis comme suit, en considérant $w_{i,j}$ comme étant la distance entre les individus o_i et o_j :

$$obj_{ICS} = \min \sum_{(i,j)\in[I]} w_{i,j} u_{i,j} \tag{2.15}$$

$$obj_{ICD} = \max \sum_{(i,j)\in[I]} w_{i,j}(1 - u_{i,j}) \tag{2.16}$$

Dans certains cas, les propriétés des clusters sont étudiées comme objectif : par exemple dans [Ouali et al., 2016b], les auteurs peuvent chercher à maximiser la taille des clusters choisis (parmi les possibles candidats).

Contraindre le clustering par l'optimisation

La modélisation du clustering par un PLNE facilitant l'expression des contraintes sur les groupes recherchés, le clustering sous contraintes a été largement modélisé de cette manière [Mueller and Kramer, 2010]. Parmi les contraintes supplémentaires pouvant être modélisées, nous pouvons citer :

— *must-link* : force l'assignation de deux objets à un même cluster :

$$y_{i,k} = y_{j,k} \ \forall k \tag{2.17}$$

— *cannot-link* : interdit l'assignation de deux objets à un même cluster :

$$y_{i,k} + y_{j,k} \leq 1 \ \forall k \tag{2.18}$$

— *margin-min* : force l'assignation de deux objets à un même cluster sous condition d'une distance D :

$$w_{i,j} < D \rightarrow must - link(i,j) \tag{2.19}$$

— *diameter-max* : interdit l'assignation de deux objets à un même cluster sous condition d'une distance D :

$$w_{i,j} > D \rightarrow cannot - link(i,j) \tag{2.20}$$

— *capacity-max* : contraint la taille des clusters :

$$\sum_{i} y_{i,k} \leq \beta \tag{2.21}$$

Ces contraintes reflètent souvent la connaissance du domaine d'application du clustering et sont donc très spécifiques aux cas d'usage. Dans notre formulation du clustering, nous utilisons ce type de contraintes supplémentaires afin de gérer la dynamicité de notre problème de clustering.

Le dialogue clustering dynamique - optimisation

La modélisation du problème de clustering par un problème d'optimisation, constituant une première interaction entre le domaine de l'optimisation et celui de l'apprentissage non supervisé, a été très étudiée. Cependant, nous utilisons dans cette thèse cette modélisation dans le cadre de données évolutives, et donc de clustering dynamique. Les modélisations existentes de clustering par des problèmes d'optimisation ne se placent pas dans ce cadre.

De plus, comme évoqué en Section 2.2, nous utilisons le clustering pour contraindre notre stratégie de placement, en utilisant les profils ainsi identifiés. Dans cette optique, nous nous intéressons à l'utilisation d'une modélisation du clustering dans le but d'exprimer la modélisation du placement par la suite (voir Section 3.2).

En Section 1.2, nous avons discuté des liens existants entre les domaines de l'apprentissage et de l'optimisation. Nous avons remarqué que les techniques existantes d'un domaine sont principalement utilisées pour l'amélioration ou l'accélération des tech-

niques de l'autre. Or, à travers notre méthode, nous cherchons une interaction et une utilisation combinée des deux, qui s'utiliseraient mutuellement.

Dans cette thèse, cette interaction se matérialise à plusieurs niveaux : l'élaboration d'une stratégie de placement utilisant des profils de consommation, la modélisation du problème de clustering de profils de consommation par un problème d'optimisation, l'utilisation du résultat de clustering dans la modélisation du problème de placement, et l'utilisation de techniques d'optimisation pour mettre à jour le clustering (et le placement associé) suivant l'évolution des données.

2.3 Conclusion et discussions

Dans cette thèse, nous nous intéressons au problème d'allocation de ressources pour conteneurs, plus particulièrement au problème de placement de ces derniers.

Alors que la plupart des travaux attaquant le problème d'allocation de ressources proposent soit des méthodes de placement initial, sans modification dans le temps, soit des méthodes d'ajustement des niveaux de ressources allouées sans remettre en cause le placement, nous proposons une solution de placement évolutive en fonction de l'évolution des consommations des conteneurs.

Pour cela, nous établissons une solution de placement basée sur les profils de consommation des conteneurs. Les méthodes de placement de micro-services n'utilisent pas la notion de profil de consommation, mais plutôt des techniques *spread*, de *binpacking* ou encore l'action directe d'un administrateur système.

L'identification des profils de consommation se fait grâce au clustering, un problème d'apprentissage non supervisé largement étudié dans de nombreux contextes. En particulier, le clustering de séries temporelles, représentant l'évolution de consommation des ressources dans notre cas, est souvent calculé grâce à des méthodes classiques de clustering (e.g. *K-means*), en utilisant soit une distance usuelle (e.g. distance euclidienne) permettant une comparaison de l'aspect temporel des séries, soit une distance adaptée aux séries temporelles (e.g. DTW) qui permet une comparaison axée sur les formes des séries. Avec cette dernière distance, des séries temporelles présentant des formes identiques mais décalées dans le temps seraient identifiées comme deux profils similaires, or nous souhaitons à l'inverse privilégier l'aspect temporel des profils, et donc pénaliser ceux en opposition de phase. C'est pour cette raison que nous utilisons la distance euclidienne. Par ailleurs, bien que plusieurs méthodes aient été implémentées (voir Section 3.3), nous utilisons l'algorithme *K-means* pour sa faible complexité comparée à d'autres méthodes, pour les caractéristiques des clusters formés (pas d'équilibrage en terme de taille) ainsi que pour le caractère itératif de son implémentation, adapté au contexte de *streaming* de notre problématique.

Ce contexte de *streaming*, faisant évoluer en continue les ressources effectivement consommées par les conteneurs, nous amène à remettre en cause ces solutions de clustering et de placement dans le temps. La modélisation de ces problèmes par des problèmes d'optimisation, combinée à la notion de dualité, nous permet de répondre à cette problématique. Bien que les problèmes de clustering et d'allocation de ressources aient déjà été modélisés par des problèmes d'optimisation, la remise en cause dans le temps de leur solution grâce à ces modélisations constitue une nouveauté.

Chapitre 3

Placement et clustering évolutifs

Sommaire

Cette deuxième partie a pour but de détailler tous les aspects de notre méthode, alliant apprentissage non supervisé et optimisation à travers des boucles.

Rappelons que le problème considéré est l'allocation de ressources en contexte conteneurs, pour lesquels nous ne connaissons pas à l'avance les consommations de ressources. La première difficulté est donc de proposer des solutions de placement ajustables dans le temps afin de garantir la qualité de la solution de ce placement malgré l'évolution des consommations. La qualité de cette solution dépend de plusieurs critères relatifs au métier et qui sont, pour certains, contradictoires. Par exemple, la minimisation du nombre de serveurs utilisés est un objectif classique souvent recherché, mais les pics (ou creux) de consommations sont des événements que l'on cherche à éviter. Or il est souvent difficile de garantir une plus grande stabilité de consommation sur un nombre d'hôtes plus restreint. Nous avons donc un problème dynamique pour lequel la qualité des solutions est difficilement quantifiable étant donnée la dimension multi-objectifs du problème. Pour cela, nous utilisons un profilage des individus considérés, obtenu grâce à un clustering, afin de guider la solution de placement. Ce clustering doit donc, tout comme le placement, être maintenu à jour dans le temps en fonction de l'évolution des profils de consommation. L'évolution du clustering doit guider l'évolution du placement, sans considérer un replacement intégral étant donné l'aspect *online* du problème.

Dans notre méthode complète, une première phase est observée pour obtenir des premières solutions. Cette phase est détaillée en Section 3.3, avant de détailler en Section 3.4 le fonctionnement des boucles pour mettre à jour (ou valider) les solutions résultant de la première phase. Tout au long des descriptions, nous utilisons un ensemble de notations dont un récapitulatif se trouve dans le Tableau 3.1.

Notation	Signification
I	Nombre d'individus (conteneurs)
M	Nombre de nœuds
K	Nombre de clusters (clustering)
T_{total}	Temps total considéré
T_{obs}	Temps d'initialisation
T_{run}	Temps d'exécution de la boucle
$\{o_1, o_2, \ldots, o_I\}$	Ensemble des individus
$\{n_1, n_2, \ldots, n_M\}$	Ensemble des nœuds
$\{c_1, c_2, \ldots, c_K\}$	Ensemble des clusters
D_t	Données concernant le temps t (entre $t-\tau$ et t)
τ	Taille des fenêtres de temps
$tick$	Taille du pas pour avancer dans le temps
VM	Machine Virtuelle

TABLEAU 3.1 – Index des notations utilisées dans le Chapitre 3

3.1 Informations préliminaires

3.1.1 Notations et gestion du temps

Afin de décrire notre méthode, ainsi que pour l'évaluer, nous utilisons un jeu de données représentant l'évolution de consommations de ressources d'un ensemble de conteneurs, appelés $\{o_1, o_2, \ldots, o_I\}$ - I étant le nombre de conteneurs représentés dans le jeu de données, initialement placés sur un ensemble de serveurs, appelés $\{n_1, n_2, \ldots, n_M\}$ - M étant le nombre de serveurs présents dans le jeu de données. Un exemple de jeu de données représentées sous forme de séries temporelles se trouve en Figure 3.1. Ces données seront utilisées tout au long de cette partie pour illustrer les différentes étapes de la méthode.

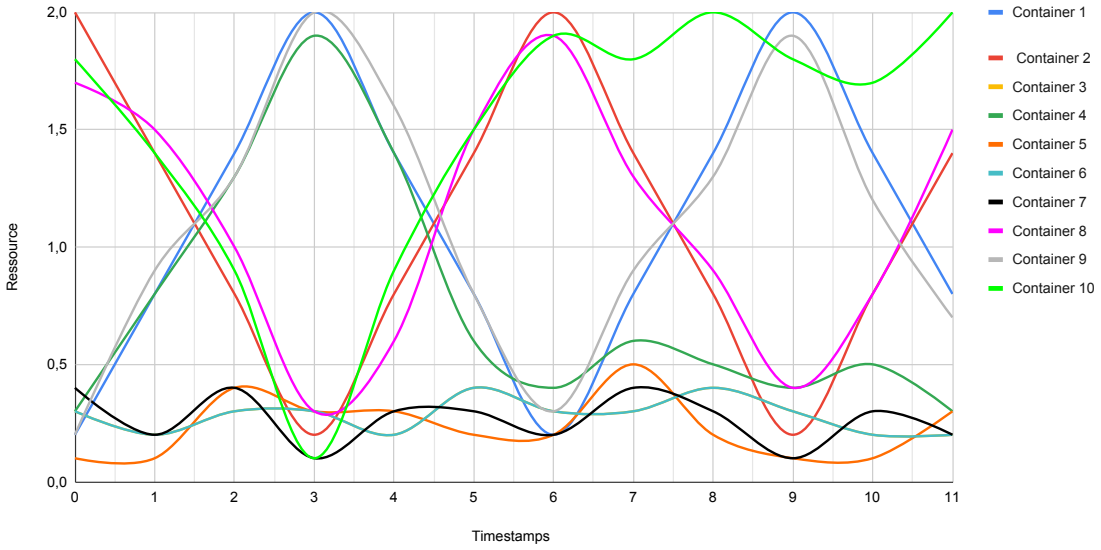


FIGURE 3.1 – Consommation d'une ressource de 10 conteneurs sur 12 points de données.

Comme pour tout problème dynamique, la notion de temps est très importante pour la compréhension de notre méthode. C'est pourquoi nous détaillons ici comment il est géré au sein de notre méthode, tout en introduisant les notations importantes.

— **Identification de deux périodes** : l'axe principale de notre méthode est une boucle, dont l'objectif est de maintenir la qualité de la solution de placement à l'arrivée de nouvelles données. Celle-ci fonctionnant avec des solutions (de clustering et de placement) existantes à chaque itération, ces solutions ont besoin d'être initialisées. C'est pourquoi nous utilisons une période d'observation, appelée T_{obs}, afin d'obtenir une première solution de clustering, qui est ensuite utilisée pour obtenir une première solution de placement. La période sur laquelle la boucle est exécutée est appelée période d'exploitation et notée T_{run}. L'ensemble des actions et calculs menés pendant ces deux périodes T_{obs} et T_{run} sont respectivement décrits en Section 3.3 et en Section 3.4.

— **Fenêtres temporelles de travail** : la gestion du temps se fait par fenêtres temporelles recouvrantes, de longueur τ. Ceci signifie qu'à tout temps t, nous travaillons avec des données D_t, comprenant les données de consommations comprises entre les temps $t - \tau$ et t. Pour chaque individu, ces données, considérées comme une série temporelle, constitue son profil de consommation sur la période $[t - \tau; t]$.

— **La progression dans le temps** : Pour avancer dans le temps, nous utilisons un paramètre *tick* qui définit le nombre de points de données ajoutés avant d'exécuter une nouvelle boucle. Par exemple, si nous nous plaçons au temps t avec $tick = 2$, la prochaine boucle se fera au temps $t + 2$ en considérant donc les données comprises entre les temps $t - \tau + 2$ et $t + 2$.

Il est important de noter que la durée de la période d'observation T_{obs}, la durée de la période d'exploitation T_{run} et la longueur des différentes fenêtres temporelles τ sont totalement indépendantes : T_{obs} peut par exemple contenir plusieurs τ.

La gestion du temps et des fenêtres temporelles est illustrée sur les données d'exemple en Figure 3.2.

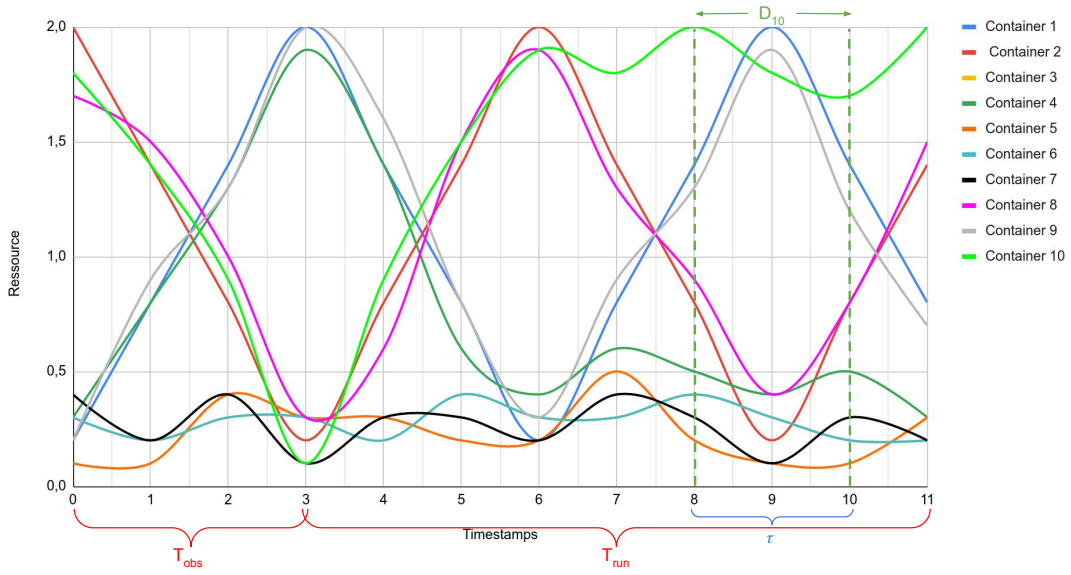


FIGURE 3.2 – Exemple de périodes et de fenêtres temporelles, avec $\tau = 3$.

3.1.2 Méthodologie complète

Avant d'entrer dans les détails de chaque étape de la méthode, et notamment de la boucle, nous donnons une vision globale de son fonctionnement.

Inputs de la boucle

En exécutant la boucle, nous avons à tout temps t :

— une solution de clustering C_{t-1}, issu de la résolution précédente ou d'un clustering externe ;

— une solution de placement A_{t-1} issu de la résolution précédente ou d'un placement externe ;

— les données de consommation D_t comprises entre $t - \tau$ et t ;

— les valeurs des variables duales $\overline{u_{ij(t-1)}}$ et $\overline{v_{ij(t-1)}}$ associées respectivement aux contraintes $mustLink_C$ et $mustLink_A$ représentant les solutions C_{t-1} et A_{t-1}.

Les solutions C_{t-1} et A_{t-1} sont le résultat de l'itération précédente $(t-1)$ de la boucle (ou de la phase d'initialisation si la boucle actuelle est la première itération).

Processus global de la boucle

Afin d'évaluer à quel point les solutions actuelles C_{t-1} et A_{t-1} doivent être adaptées au vu des nouvelles données D_t, les problèmes de clustering et d'affectation sont modélisés comme des problèmes linéaires en nombres entiers (PLNE, ou ILP en anglais), appelés respectivement $\mathcal{CP}$ et $\mathcal{AP}$. Ces problèmes d'optimisation sont détaillés en Section 3.2. Ces derniers sont en fait utilisés, avec l'information des solutions courantes et compte tenu des relaxations continues (respectivement $\mathcal{CCP}$ et $\mathcal{CAP}$), pour évaluer ces solutions selon un objectif adapté, grâce aux valeurs des variables duales associées à des contraintes spécifiquement ajoutées. Les grandes étapes de la boucle sont donc les suivantes :

1. Evaluer le clustering actuel C_{t-1} sur D_t ;
2. Modifier le clustering C_{t-1} en conséquence pour obtenir un nouveau clustering C_t ;
3. Evaluer le placement actuel A_{t-1} avec le nouveau clustering C_t ;
4. Modifier le placement A_{t-1} en conséquence pour obtenir un nouveau placement A_t.

Chacune de ces étapes est détaillée dans les sections suivantes. Un schéma représentant le processus global de la boucle est visible en Figure 3.3.

3.2 Description des modèles d'optimisation

Au sein de notre système de boucles, les problèmes de clustering et de placement sont modélisés par des problèmes d'optimisation (un modèle pour chaque problème) qui sont dépendants entre eux (la solution de l'un contraint l'autre). La construction de ces modèles est détaillée ici, avant de détailler comment ils sont utilisés dans notre méthode (voir Section 3.4). Un récapitulatif des notations utilisées dans la description des modèles d'optimisation se trouve en Tableau 3.2.

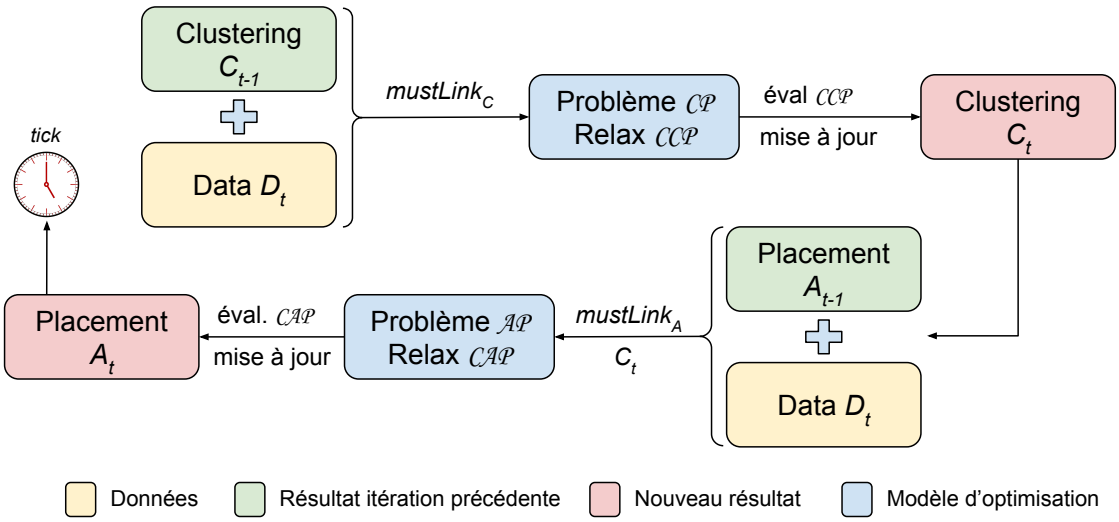


FIGURE 3.3 – Schéma global du fonctionnement de la boucle.

Notation	Signification
I	Nombre d'individus (conteneurs)
M	Nombre de nœuds
K	Nombre de clusters (clustering)
T	Temps total considéré
$\{o_1, o_2, \ldots, o_I\}$	Ensemble des individus
$\{n_1, n_2, \ldots, n_M\}$	Ensemble des nœuds
$\{c_1, c_2, \ldots, c_K\}$	Ensemble des clusters
VM	Machine Virtuelle
Couleur	**Signification**
Vert	Données
Bleu	Variables de décision
Rouge	Paramètres

TABLEAU 3.2 – Index des notations pour les modèles d'optimisation

.

3.2.1 Formulation du problème de clustering

Nous détaillons dans un premier temps la formulation de notre problème de clustering. La modélisation de ce problème a été discutée en Section 2.2.2. Ce qui nous intéresse à travers cette modélisation, c'est sa mise à jour dans le temps et son utilisation pour le placement. L'originalité ici n'est donc pas cette modélisation mais son utilisation.

Comme principale variable de décision, liant deux individus o_i et o_j , nous utilisons la variable $u_{i,j}$ rappelée ci-dessous :

$$u_{i,j} = \begin{cases} 1 & \text{si } o_i \text{ et } o_j \text{ sont dans le même cluster} \\ 0 & \text{sinon} \end{cases}$$

Cependant, afin de représenter les clusters et faciliter notamment la formalisation de certaines contraintes, nous utilisons également la variable $y_{i,k}$, liant l'individu o_i et le cluster c_k et définie ci-dessous :

$$y_{i,k} = \begin{cases} 1 & \text{si le conteneur } o_i \text{ appartient au cluster } c_k \\ 0 & \text{sinon} \end{cases}$$

Afin de garantir que chaque conteneur est affecté à exactement un cluster, les contraintes d'assignation suivantes sont ajoutées :

$$\sum_{k=1}^{K} y_{i,k} = 1 \ \forall \ i \in [1..I] \tag{3.1}$$

Pour assurer la cohérence de la variable $u_{i,j}$, nous ajoutons les contraintes :

$$u_{i,j} = \sum_{k=1}^{K} y_{i,k} y_{j,k} \tag{3.2}$$

Pour savoir si le cluster c_k est "actif" (contient des conteneurs), on utilise la variable suivante :

$$b_k = \begin{cases} 1 & \text{si au moins un conteneur est dans le cluster } c_k \\ 0 & \text{sinon} \end{cases}$$

Pour s'assurer que cette dernière variable est bien à 1 quand nécessaire, nous utilisons la contrainte suivante :

$$y_{i,k} \leq b_k \ \forall \ i \in [1..I], k \in [1..K] \tag{3.3}$$

Ensuite afin de fixer le nombre de clusters, nous ajoutons la contrainte :

$$\sum_{k=1}^{K} b_k = nb\_clusters \tag{3.4}$$

Il est intéressant de noter que cette contrainte peut être modifiée en cas par exemple d'ajustement du nombre de clusters. En effet, nous pouvons utiliser $nb\_clusters_{min}$ et $nb\_clusters_{max}$ respectivement comme une borne minimale et maximale sur le nombre de clusters à utiliser et ainsi utiliser une contrainte du type :

$$nb\_clusters_{min} \leq \sum_{k=1}^{K} b_k \leq nb\_clusters_{max} \tag{3.5}$$

Comme énoncé précédemment, l'objectif couramment optimisé dans le clustering est la minimisation de la dissimilarité intra-clusters, rappelé en Equation 3.6 :

$$obj_{ICS} = \min \sum_{(i,j)\in[1..I]} w_{i,j}u_{i,j} \tag{3.6}$$

Finalement, notre modèle de clustering complet est le suivant :

$$(\mathcal{CP}) \quad \begin{array}{lll} \text{minimize} & \sum_{(i,j)\in[1..I]} w_{i,j}u_{i,j} & \\ \text{subject to} & \sum_{k=1}^{K} y_{i,k} = 1 & \forall i \in [1..I] \\ & y_{i,k} \leq b_k & \forall i \in [1..I], k \in [1..K] \\ & \sum_{k=1}^{K} b_k \leq nb\_clusters & \\ & u_{i,j} = \sum_{k=1}^{K} y_{i,k}y_{j,k} & \forall (i,j) \in [1..I] \end{array}$$

La modélisation du problème de clustering via l'optimisation permet une première interaction entre ce dernier et l'apprentissage. Cette interaction est renforcée par l'idée d'utiliser le résultat de clustering dans la formulation du problème de placement. Nous avons donc une méthode à deux niveaux :

1. un problème modélisant le clustering utilisé pour mettre à jour la solution de clustering courante en fonction des nouvelles données ;
2. un problème modélisant le placement, utilisant les nouvelles données et la nouvelle solution de clustering (trouvée grâce au précédent modèle) pour mettre à jour la solution de placement courante.

Nous détaillons donc par la suite comment nous modélisons notre problème de placement, et comment le clustering est utilisé dans cette modélisation.

3.2.2 Formulation de notre placement

Comme nous l'avons détaillé en Section 2.1.1, le problème de placement de conteneurs n'a jusqu'à maintenant pas été modélisé par un problème d'optimisation. Pour cela, nous utilisons le même type de formulation que pour un problème de placement de machines virtuelles, que nous détaillons ci-dessous.

Par ailleurs, nous utilisons le résultat du clustering, décrit ci-dessus, pour décrire la fonction objectif de notre modèle de placement.

Soit $o_i, i \in [1..I]$ notre ensemble de conteneurs et $n_m, m \in [1..M]$ notre ensemble de nœuds. On commence par créer une variable $x_{i,m}$ de taille $I \times M$ telle que :

$$x_{i,m} = \begin{cases} 1 & \text{si le conteneur } o_i \text{ est placé sur le nœud } n_m \\ 0 & \text{sinon} \end{cases}$$

Pour s'assurer que chaque conteneur est assigné exactement à un nœud, nous ajoutons la contrainte d'affectation suivante :

$$\sum_{m=1}^{M} x_{i,m} = 1 \ \forall \ i \in [1..I] \tag{3.7}$$

Soit $cons_{m,t}$ la consommation totale sur un nœud n_m à l'instant t. Nous pouvons formuler, pour chaque $m \in [1..M]$, $cons_{m,t}$ en fonction des variables $x_{i,m}$ et des données de consommation de chaque conteneur o_i au temps t, appelées $cons_{i,t}$:

$$cons_{m,t} = \sum_{i=1}^{I} x_{i,m} cons_{i,t} \tag{3.8}$$

Ensuite pour s'assurer que la consommation totale des nœuds ne dépasse pas leur capacité, appelée cap_m, nous ajoutons les contraintes de capacité :

$$cons_{m,t} \leq cap_m \ \forall \ m \in [1..M], t \in [1..T] \tag{3.9}$$

La variable suivante est ajoutée pour limiter le nombre de serveurs utilisés :

$$a_m = \begin{cases} 1 & \text{si le nœud } n_m \text{ est allumé (au moins 1 conteneur dessus)} \\ 0 & \text{sinon} \end{cases}$$

Pour forcer a_m à 1 si au moins 1 conteneur est placé sur le serveur n_m, la contrainte suivante est ajoutée :

$$x_{i,m} \leq a_m \ \forall \ i \in [1..I], m \in [1..M] \tag{3.10}$$

Pour borner le nombre de serveurs utilisés, en fonction de l'infrastructure fournissant le nombre maximum de serveurs max_nodes, nous utilisons la contrainte :

$$\sum_{m=1}^{M} a_m \leq max\_nodes \tag{3.11}$$

Fonction objectif

Les objectifs du placement sont multiples et répondent à des besoins métier. Par exemple dans [N. Vhatkar and Bhole, 2019], les auteurs cherchent à minimiser la somme de plusieurs critères (l'équilibrage des nœuds, les pannes systèmes, la distance totale du réseau) spécifiques à leurs besoins. Dans [Rezvani et al., 2014], les auteurs cherchent à maximiser l'utilisation CPU des machines. Cet objectif est propre au placement de machines virtuelles à tailles fixes et ne s'applique pas à notre contexte où l'utilisation CPU n'est pas fixe. Dans notre cas, le premier objectif, qui peut également l'être en contexte de VMs, peut être de minimiser le nombre de nœuds utilisés :

$$obj_{noeuds} = \min \sum_{m \in [1..M]} a_m \tag{3.12}$$

Par ailleurs, nous souhaitons également maximiser la stabilité de consommation sur les nœuds, pour ainsi mieux anticiper les changements de régime et minimiser les incidents de production. Nous avons plusieurs moyens d'exprimer la stabilité de consommation sur les nœuds :

1. L'amplitude maximale de consommation. Pour chaque nœud, son amplitude de consommation s'écrit :

$$amp_m = \max_{\forall t \in [1..T]} (cons_{m,t}) - \min_{\forall t \in [1..T]} (cons_{m,t}) \tag{3.13}$$

   Nous pouvons ensuite utiliser l'amplitude maximale sur tous les nœuds comme objectif à minimiser :

$$obj_{amp} = \min_{\forall m \in [1..M]} (amp_m) \tag{3.14}$$

2. La variance de consommation. Pour chaque nœud, sa variance de consommation s'écrit :

$$var_m = \frac{1}{T} \sum_{t=1}^{T} (cons_{m,t} - \overline{cons_m})^2 \tag{3.15}$$

   Avec $\overline{cons_m}$ la moyenne de consommation sur le nœud n_m, qui s'écrit :

$$\overline{cons_m} = \frac{1}{T} \sum_{t=1}^{T} cons_{m,t} \tag{3.16}$$

   Nous pouvons ensuite utiliser la variance maximale sur l'ensemble des nœuds N

comme objectif à minimiser :

$$obj_{var} = \min_{\forall m \in [1..M]} (var_m) \tag{3.17}$$

Utilisation du clustering

Comme mentionné précédemment, l'utilisation du résultat de clustering dans le problème de placement permet une interaction forte entre les profils de consommation extraits par le clustering et le placement. Comme nous cherchons à *stabiliser* le plus possible les consommations totales de ressources sur les nœuds, l'utilisation des profils de consommation pour le placement est intéressant. En effet, si nous plaçons sur le même nœud deux individus aux profils similaires, la consommation totale sur ce nœud n'en sera que moins stable (sauf si leurs profils sont eux-même stables). À l'inverse, la co-localisation de conteneurs aux profils *différents* facilitera la stabilité de consommation sur le nœud en question.

C'est pourquoi nous récupérons la solution de clustering via les variables de décision $u_{i,j}$, qui sont donc maintenant des données. Afin d'être utilisée conjointement avec les données $u_{i,j}$, nous ajoutons la variable suivante, de taille $I \times I$ pour chaque paire (o_i, o_j) de conteneurs :

$$v_{i,j} = \begin{cases} 1 & \text{si } o_i \text{ et } o_j \text{ sont sur le même nœud} \\ 0 & \text{sinon} \end{cases}$$

Pour garantir cette égalité, on ajoute les contraintes suivantes :

$$v_{i,j} = \sum_{m=1}^{M} x_{i,m} x_{j,m} \ \forall \ (i,j) \in [1..I] \tag{3.18}$$

Enfin, nous considérons une fonction objectif basée sur des profils issus du clustering. Comme justifié ci-dessus, nous souhaitons éviter la *co-localisation* de conteneurs ayant des profils similaires, en évitant que le produit $u_{i,j}v_{i,j}$ soit égal à 1. Par conséquent, pour les conteneurs appartenant à des clusters différents, nous souhaitons *co-localiser* les conteneurs les plus *distants*. Pour cela, nous considérons une nouvelle distance $d_{i,j}$ comme la variance de la somme des profils de o_i et o_j. En considérant $cc_{i,t}$ la valeur de consommation au temps t du profil moyen du cluster auquel appartient le conteneur o_i, et $\overline{cc_i}$ sa valeur moyenne, cette distance $d_{i,j}$ se définit comme suit :

$$d_{i,j} = \frac{1}{T} \sum_{t=1}^{T} (cc_{i,t} + cc_{j,t} - (\overline{cc_i + cc_j}))^2 \tag{3.19}$$

La *co-localisation* des conteneurs (o_i, o_j) ayant un plus petit $d_{i,j}$ est donc préférée. On

peut alors écrire la fonction objectif suivante :

$$obj_{assign} = \min \sum_{(i,j)\in[1..I]} u_{i,j}v_{i,j} + (1 - u_{i,j})v_{i,j}d_{i,j} \tag{3.20}$$

Avec cet objectif, le problème de placement entier est modélisé de la manière suivante :

$$(\mathcal{AP}) \quad \begin{array}{lll} \text{minimize} & \displaystyle\sum_{(i,j)\in[1..I]} u_{i,j}v_{i,j} + (1 - u_{i,j})v_{i,j}d_{i,j} & \\ \text{subject to} & cons_{m,t} \leq cap_m & \forall m \in [1..M],\ t \in [1..T] \\ & \displaystyle\sum_{m=1}^{M} x_{i,m} = 1 & \forall i \in [1..I] \\ & x_{i,m} \leq a_m & \forall i \in [1..I],\ m \in [1..M] \\ & \displaystyle\sum_{m=1}^{M} a_m \leq max\_nodes & \\ & \displaystyle v_{i,j} = \sum_{m=1}^{M} x_{i,m}x_{j,m} & \forall (i,j) \in [1..I] \end{array}$$

Notre contexte de streaming nous imposant l'obtention rapide de solutions, ces programmes ne sont pas la modélisation exacte de ces problèmes mais sont adaptés à une utilisation dans le but final d'adapter le placement des conteneurs aux nouvelles données de consommation de ces derniers. Des détails sur les caractéristiques de ces problèmes (nombre de variables, nombre de contraintes, temps de résolution) sont donnés en Section 4.4. L'utilisation de ces modèles est expliquée par la suite.

3.3 Initialisation de la boucle

Comme mentionné précédemment, une phase d'analyse T_{obs} est utilisée sur une partie des données d'observation (D_0), dans le but d'obtenir les premières solutions de clustering et de placement, respectivement C_0 et A_0, qui seront ensuite données en entrée aux boucles. Nous détaillons dans cette partie comment les obtenir. Pour illustrer cette première phase à partir du jeu de données d'exemple, nous choisissons de garder 4 points de données comme étant la durée de cette période d'observation (paramétrable en fonction des données utilisées). La Figure 3.4 illustre les données utilisées pendant cette période à partir de notre exemple.

Les algorithmes de clustering sont détaillés dans un premier temps, avant d'expliquer l'utilisation de son résultat pour obtenir une solution de placement.

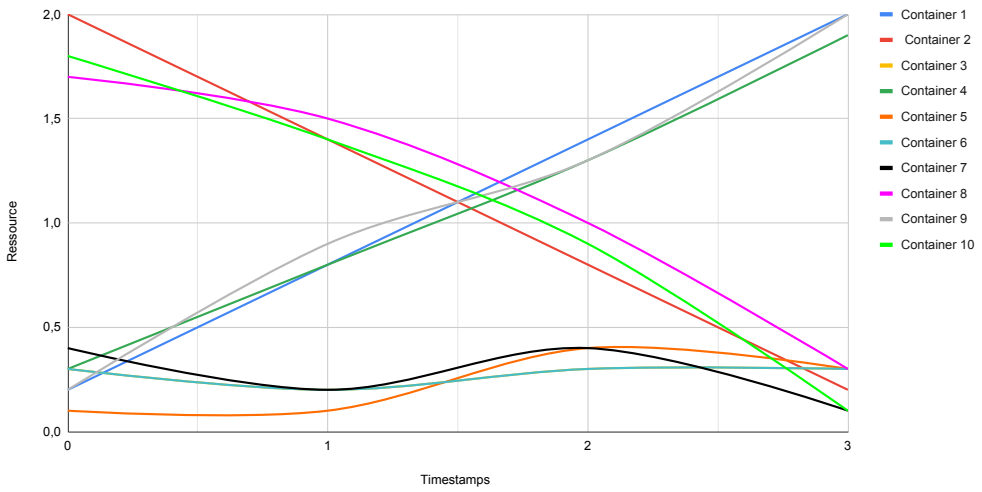


FIGURE 3.4 – Consommation d'une ressource de 10 conteneurs sur une période d'analyse de 4 points de données.

3.3.1 Clustering initial

Le clustering est utilisé dans notre cas pour identifier des profils de consommation de ressources, grâce aux traces de consommation passées. Comme détaillé en Section 2.1.2, il existe beaucoup de travaux et de techniques différentes pour obtenir un clustering, y compris pour les séries temporelles que nous avons également étudiées dans la même section.

Dans un premier temps, nous avions vu que l'une des distances les plus utilisées pour le clustering de séries temporelles était le *Dynamic Time Warping (DTW)*. Cependant, cette distance favorisant le regroupement d'individus présentant des formes similaires sans considérer la différence temporelle de ces formes (par exemple deux séries temporelles présentant un niveau de consommation équivalent et le même pic de consommation à des moments différents pourraient être regroupées ensemble). Dans notre cas, nous souhaitons regrouper les individus ayant les mêmes profils de consommation en même temps, et justement éviter de colocaliser des individus ayant des pics de consommation au même moment. C'est pour cela que nous nous sommes intéressés à la distance euclidienne.

La distance euclidienne est très adaptée à l'utilisation de méthodes basées sur les caractéristiques, d'autant plus qu'une comparaison point à point des séries temporelles nous permet plus facilement de séparer des individus présentant des niveaux de consommation particulièrement élevés (ou particulièrement bas) à des moments différents. La méthode la plus utilisée de clustering basées sur les caractéristiques est le *K-means* [MacQueen, 1967b], dont le fonctionnement et les raisons pour lesquelles nous choisissons cet algorithme sont détaillés en Section 2.1.2.

L'implémentation de cet algorithme se fait en utilisant le module *scikit-learn* [Pedregosa

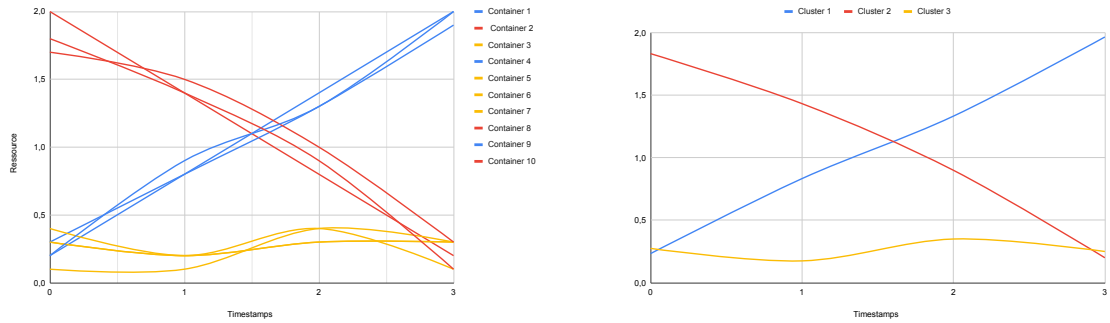
et al., 2011]. L'algorithme nécessite la fourniture de certains paramètres, en particulier le nombre de clusters (k) à utiliser. Bien que la fourniture de ce paramètre soit laissée aux soins de l'utilisateur, nous proposons une méthode pour obtenir une valeur de k à utiliser. Cette méthode utilise le *Gap statistic* et est détaillée en Section 4.2.

L'initialisation des graines se fait en suivant la méthode *K-means++* [Arthur and Vassilvitskii, 2006], dont l'idée principale est de choisir des graines initiales les plus éloignées possibles. Les principales étapes de cette initialisation sont décrites en Algorithme 4.

Algorithme 4 : Initialisation des graines pour *K-means* en utilisant *K-means++*

1 Choisir aléatoirement un centre (une graine) parmi les points de données
2 **while** *k centres ne sont pas calculés* **do**
3 Pour chaque point de données non considéré, calculer $D(x)$ la distance entre x et le centre déjà choisi le plus proche
4 Choisir aléatoirement un nouveau point de données comme centre, en utilisant une distribution de probabilité pondérée où x est choisi avec une probabilité proportionnelle à $D(x)$
5 Exécuter l'algorithme *K-means* classique à partir des k centres calculés

Le résultat du clustering C_0 sur les données d'exemples se trouve en Figure 3.5.



(a) Profils des conteneurs clusterisés

(b) Profils moyens des clusters

FIGURE 3.5 – Résultat du clustering C_0 avec $k = 3$ pour les données d'exemples.

3.3.2 Placement initial

Ces profils de consommation sont utilisés pour obtenir une solution de placement de micro-services optimisée dans le temps. Un algorithme a donc été développé dans le but d'obtenir des consommations totales sur les nœuds les plus stables possibles, à partir des résultats de clustering.

Le clustering produit des groupes de conteneurs ayant des profils de consommation de ressources similaires. Les informations sont ensuite utilisées pour alimenter une heuristique de placement, dont l'objectif principal est de co-localiser les conteneurs avec des profils distants, afin d'avoir la consommation totale la plus stable sur les nœuds et le nombre de nœuds utilisés le plus petit possible.

L'heuristique utilisée est basée sur l'idée de colocaliser en priorité les individus présentant une plus grande symétrie dans leurs profils : les paires de clusters sont ordonnées par variance croissante de la somme des profils, puis les conteneurs appartenant aux clusters de la première paire sont co-localisés, avant de considérer les deux prochains clusters de la liste (sans retraiter un cluster déjà traité via une autre paire auparavant). La description de cet algorithme est présent en Algorithme 5. Cet algorithme nous donne notre première solution de placement A_0.

Algorithme 5 : Heuristique de placement *cluster pairwise*

Input : Clustering $C_0 \to K$ clusters
Output : Placement $A_0 \to I$ Conteneurs placés sur M Nœuds ($I \gg M$)
Data : Capacités des nœuds cap_m, consommations des conteneurs $conso_{o_{i,t}}$

```
1  Calculer profils moyens c_a, 1 ≤ a ≤ K
2  vs_{a,b} ← var(c_a + c_b) ∀ (a,b) ∈ 1...K, a ≠ b
3  while clusters restants > 0 do
4      Sélectionner (c_a, c_b) tels que (a,b) = argmin_{1≤a,b≤K,a≠b} vs_{a,b}
5      for o_i ∈ c_a ; o_j ∈ c_b do
6          m ← 1
7          if conso_{m,t} + conso_{o_i,t} + conso_{o_j,t} < cap_m then
8              Placer o_i et o_j sur n_m
9          else
10             m++
11     Supprimer clusters c_a et c_b
12 if clusters restants = 1 then                          ▷ spread conteneurs
13     c_k ← cluster restant
14     m ← 1
15     forall o_i ∈ c_k do
16         if conso_{m,t} + conso_{o_i,t} < cap_m then
17             Placer o_i sur n_m
18         else
19             m++
20         m++
```

3.3.2.1 Exemple d'utilisation de l'heuristique

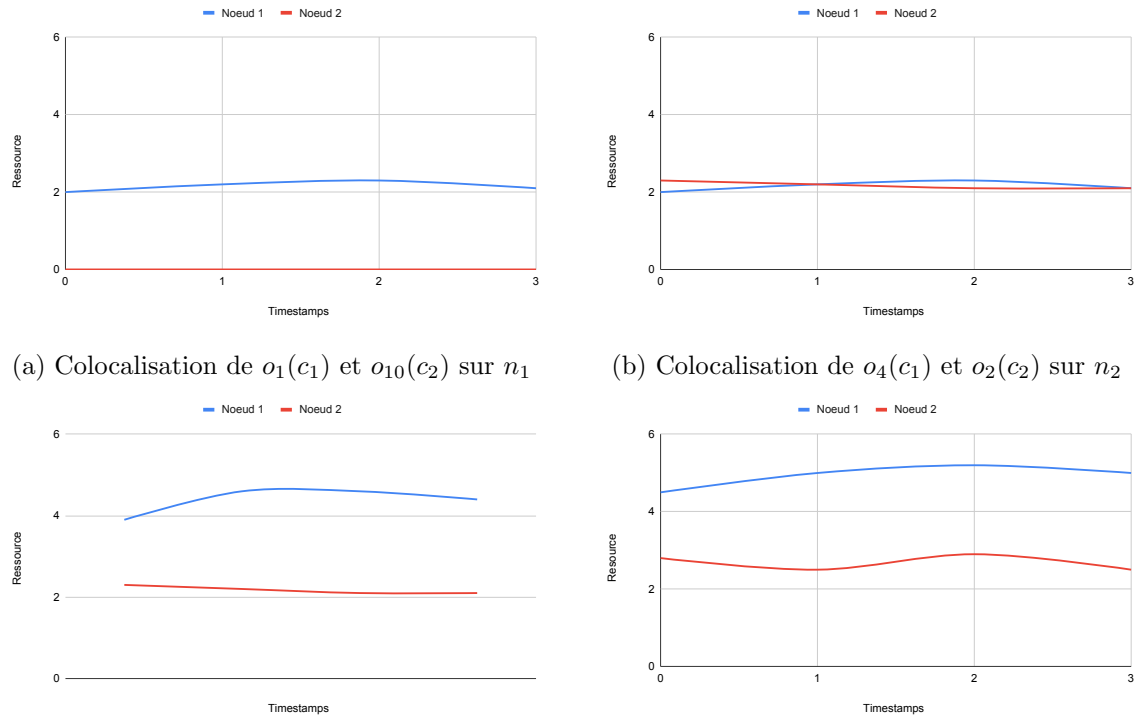
Pour illustrer le fonctionnement de notre heuristique, nous utilisons le résultat de clustering C_0 d'exemple en Figure 3.5. Considérons 2 serveurs de capacité maximale 6 sur une certaine ressource. La variance de la somme de chaque paire de clusters est indiquée dans le Tableau 3.3.

D'après ces valeurs, les clusters c_1 et c_2 sont considérés en premiers, la variance de la somme de leur profil étant la plus petite. Nous colocalisons donc leurs individus respectifs par paire, en itérant sur les 2 nœuds disponibles. L'ordre de sélection des

Clusters	c_1	c_2	c_3
c_1	0.86	0.002	0.24
c_2	0.002	1.02	0.23
c_3	0.24	0.23	0.02

TABLEAU 3.3 – Tableau de la variance de la somme des profils moyens des clusters.

individus au sein d'un cluster est aléatoire : ici les premiers à être considérés sont les conteneurs o_1 (appartenant au cluster c_1) et o_{10} (appartenant au cluster c_2), placés sur le nœud n_1. La paire suivante ($o_4(c_1)$ et $o_2(c_2)$) est placé sur le nœud n_2, tandis que nous revenons sur le nœud n_1 pour la dernière paire ($o_9(c_1)$ et $o_8(c_2)$). Le cluster c_3 étant le dernier cluster à considérer, ses conteneurs sont répartis sur les nœuds suivant un spread classique : o_3 sur n_1, o_5 sur n_2, o_6 sur n_1 puis o_7 sur n_2. L'évolution de consommation de ressource sur les nœuds n_1 et n_2, au fil du placement des conteneurs, est illustré en Figure 3.6.



(a) Colocalisation de $o_1(c_1)$ et $o_{10}(c_2)$ sur n_1

(b) Colocalisation de $o_4(c_1)$ et $o_2(c_2)$ sur n_2

(c) Colocalisation de $o_9(c_1)$ et $o_8(c_2)$ sur n_1

(d) Répartition de $o_3, o_5, o_6, o_7(c_3)$ sur n_1 et n_2

FIGURE 3.6 – Evolution des consommations des nœuds n_1 et n_2 pendant la construction du placement A_0.

Nous avons décrit ici comment les solutions initiales, servant d'entrée pour les boucles, sont obtenues.

L'algorithme *K-means* est utilisé pour obtenir un premier clustering, dont le résultat est utilisé dans le développement d'une heuristique originale utilisant un système de *bin-packing* adapté à la prise en compte des profils identifiés par le clustering.

Ces méthodes sont peu coûteuses, et la performance des solutions obtenues n'est pas garantie. Mais ces solutions sont amenées à être améliorées et/ou adaptées dans le temps grâce aux boucles. Cette mise à jour dans le temps est décrite par la suite.

3.4 Ajustement des solutions dans le temps

Lorsque nous exécutons notre boucle à un temps t, l'objectif principal est d'évaluer la pertinence de la solution courante de placement, suivant l'arrivée de nouvelles données de consommation D_t, et de modifier ce placement si nécessaire.

Comme nous utilisons le clustering (via les profils de consommation) pour guider le placement, nous cherchons dans un premier temps à évaluer ce profilage des individus. C'est la mise à jour (ou non) de notre clustering qui va impacter l'évaluation du placement courant, et donc sa modification (ou validation).

Afin de faciliter la compréhension de cette boucle, nous listons dans un premier temps les entrées de celle-ci, avant de détailler le processus global suivi pendant la boucle. Les modes de déclenchement de la boucle sont enfin expliqués.

3.4.1 Mise à jour du clustering $C_{t-1} \rightarrow C_t$

Avant de détailler la méthode de mise à jour du clustering, nous rappelons les notations utilisées dans cette partie en Tableau 3.4.

Notation	Signification
I	Nombre d'individus (conteneurs)
K	Nombre de clusters (clustering)
$\{o_1, o_2, \ldots, o_I\}$	Ensemble des individus
$\{c_1, c_2, \ldots, c_K\}$	Ensemble des clusters
C_t	Solution de clustering au temps t
D_t	Données concernant le temps t (entre $t-\tau$ et t)
$\mathcal{CP}$	Modélisation de notre problème de clustering
$\mathcal{CCP}$	Relaxation continue du problème $\mathcal{CCP}$
$mustLink_C$	Contraintes ajoutées au problème $\mathcal{CP}$ pour fixer la solution actuelle
$\overline{u_{ijt}}$	Variables duales associées aux contraintes $mustLink_C$ au temps t
ϵ_C	Seuil de tolérance pour comparer les valeurs des variables $\overline{u_{ijt}}$
G_C	Graphe de conflit de la solution de clustering actuelle
$V_G(o_i)$	Ensemble des sommets voisins de o_i dans G_C
e_{ij} et $w(e_{ij})$	Arête liant les sommets o_i et o_j et son poids associé dans G_C
w_deg_i	Degré pondéré du sommet o_i dans G_C
L_C	Liste des individus à réassigner pour obtenir C_t
$d_{i,k}$	Distance entre profil moyen c_k et o_i

TABLEAU 3.4 – Index des notations utilisées pour le clustering.

Pour mettre à jour le clustering, nous gardons le même nombre de clusters K et réassignons les conteneurs déviant de leur cluster courant vers un cluster plus proche. Pour détecter ces conteneurs nous utilisons un système de conflits de paires de conteneurs, afin de les réassigner de faon globale et informée. La première étape de la boucle est donc la détection de ces conflits au sein de la solution de clustering C_{t-1} au vu des données D_t. Cette première étape est d'abord décrite en Algorithme 6, dans lequel nous détaillons la construction d'un graphe de conflits G_C. Ce graphe de conflit nous permet de considérer les possibles modifications d'un point de vue plus global, en considérant non pas un conflit (impliquant seulement deux individus) à la fois, mais l'ensemble des conflits en même temps pour savoir comment modifier la solution de clustering.

Algorithme 6 : Construction du graphe de conflit G_C pour le clustering

Input : Clustering C_{t-1}, Données D_t, $\overline{u_{ij(t-1)}}$, ϵ_C
Output : Graphe G_C

Construire problème $\mathcal{CP}$ avec données D_t
forall $(o_i, o_j) \in$ *même cluster* c_k *dans* C_{t-1} **do**
    Ajouter contrainte $mustLink_C$ pour o_i et o_j dans $\mathcal{CP}$
Résoudre $\mathcal{CCP}$ relaxation continue de $\mathcal{CP}$ avec données D_t
Récupérer valeurs des variables duales $\overline{u_{ijt}}$ associées aux contraintes $mustLink_C$
if $\overline{u_{ijt}} > \overline{u_{ij(t-1)}}(1 + \epsilon_C)$ **then**
    Ajouter couple (o_i, o_j) dans G_C

Pour cela, nous construisons le problème d'optimisation $\mathcal{CP}$ (voire Section 3.2), dans lequel nous fixons la solution courante C_{t-1} : nous ajoutons à $\mathcal{CP}$ les contraintes $mustLink_C$ suivantes, qui pour tout couple d'individus (o_i, o_j) appartenant au même cluster dans C_{t-1} fixe la variable correspondante $u_{i,j}$ à 1 :

$$mustLink_C : u_{i,j} = 1 \ \forall (i,j) \text{ appartenant au même cluster dans } C_{t-1} \tag{3.21}$$

Ce type de contrainte n'est pas nouveau dans la modélisation du clustering (voir Section 2.2.2) mais reflète généralement des contraintes métiers ou une connaissance a priori sur le problème étudié. Dans notre cas, nous utilisons ces contraintes pour gérer l'aspect dynamique du clustering : la remise en cause de celles-ci, par des techniques d'optimisation utilisant la notion de dualité (voir Section 2.2.2), d'une itération à une autre en considérant les nouvelles données nous permet de mettre à jour le clustering, comme nous l'expliquons par la suite.

Le problème d'optimisation $\mathcal{CP}$ modélisant le clustering, détaillé en Section 3.2, est rappelé en Figure 3.7 en y ajoutant les contraintes $mustLink_C$ décrites ci-dessus.

Au sein de ce problème $\mathcal{CP}$, les équations 3.22e entraînent la considération d'un problème quadratique. Une linéarisation de ces contraintes est appliquée, en considérant une méthodologie décrite dans [Fortet, 1960] :

$$\min \quad \sum_{(i,j)\in[1..I]} w_{i,j}u_{i,j} \tag{3.22a}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} y_{i,k} = 1 \qquad \forall i \in [1..I] \tag{3.22b}$$

$$y_{i,k} \leq b_k \qquad \forall i \in [1..I], k \in [1..K] \tag{3.22c}$$

$$\sum_{k=1}^{K} b_k \leq nb\_clusters \tag{3.22d}$$

$$u_{i,j} = \sum_{k=1}^{K} y_{i,k}y_{j,k} \qquad \forall (i,j) \in [1..I] \tag{3.22e}$$

$$u_{i,j} = 1 \qquad \forall\ (o_i, o_j) \in meme\ c_k\ dans\ C_{t-1} \tag{3.22f}$$

FIGURE 3.7 – Problème de clustering $\mathcal{CP}$ avec les contraintes $mustLink_C$.

— une nouvelle variable représentant le produit $y_{i,k}y_{j,k}$ est crée. Nous appelons cette variable $z_{i,j,k}$ qui représente l'appartenance des individus o_i et o_j au cluster c_k.

— afin de forcer l'égalité $z_{i,j,k} = y_{i,k}y_{j,k}$, l'ensemble des contraintes suivantes sont ajoutées :

$$z_{i,j,k} \leq y_{i,k} \qquad \forall i \in [1..I], k \in [1..K] \tag{3.23a}$$

$$z_{i,j,k} \leq y_{j,k} \qquad \forall j \in [1..I], k \in [1..K] \tag{3.23b}$$

$$z_{i,j,k} \geq y_{i,k} + y_{j,k} - 1 \qquad \forall (i,j) \in [1..I], k \in [1..K] \tag{3.23c}$$

$$z_{i,j,k} \geq 0 \tag{3.23d}$$

Après cette reformulation linéaire, nous considérons la relaxation continue du problème $\mathcal{CP}$, appelée $\mathcal{CCP}$. En relâchant les contraintes d'intégrité de $\mathcal{CP}$, la résolution du nouveau problème $\mathcal{CCP}$, en considérant les données actuelles D_t, est plus rapide et nous permet d'obtenir les valeurs des variables duales associées à chacune des contraintes de $\mathcal{CCP}$ (voire Section 2.2.2 pour plus de détails). Plus ces valeurs sont hautes, plus la relaxation des contraintes associées à ces variables doit nous permettre d'améliorer la solution. Ici, nous nous intéressons aux variables duales associées aux contraintes $mustLink_C$, notées $\overline{u_{ij}}$. En effet, représentant les individus regroupés dans les mêmes clusters, leur variable duale associée nous aide à détecter les individus appartenant au même cluster au temps $t-1$ et dont la distance augmente au temps t.

Pour cela, nous utilisons un seuil de tolérance appelé ϵ_C, en considérant $\overline{u_{ij(t-1)}}$ la valeur de la variable duale $\overline{u_{ij}}$ à l'itération précédente $t-1$, et sa valeur actuelle $\overline{u_{ijt}}$. Pour chaque contrainte $mustLink_C$ (Equation 3.21), si $\overline{u_{ijt}} > \overline{u_{ij(t-1)}}(1+\epsilon_C)$, alors on considère les conteneurs o_i et o_j en *conflit*. Considérer deux individus en conflit entraine donc la remise en cause de l'assignation de ces individus à un même cluster (e.g. suite un changement de profil de consommation).

L'ensemble de ces conflits nous permet de construire un graphe de conflit G_C, à partir duquel nous créons une liste d'individus à réassigner pour modifier le clustering actuel C_{t-1}. La création de cette liste est détaillée en Algorithme 7.

Algorithme 7 : Construction de la liste de réassignation L_C

Input : Clustering C_{t-1}, Données D_t, Graphe G_C
Output : Liste L_C

$L_C \leftarrow [\,]$
Trier les sommets du graphe G_C par degré décroissant
forall *Sommet* o_i *de* G_C *par degré décroissant* **do**
  if *degré*$(o_i) = 1$ **then** ▷ `cas où degré = 1`
    $o_j \leftarrow$ sommet voisin de o_i dans G_C
    $(d_{i,k}, d_{j,k}) \leftarrow$ distance entre profil moyen de c_k (auquel appartiennent o_i et o_j) et respectivement o_i et o_j
    if $d_{j,k} > d_{i,k}$ **then**
      $i = j$
    ▷ o_i `individu plus distant du profil moyen de son cluster`
  else if $\exists\ o_j$ *t.q.* *degré*$(o_j) =$ *degré*(o_i) **then** ▷ `cas où plusieurs sommets de même degré`
    $w\_deg_{max} = 0$
    forall *Sommet* o_j *t.q.* *degré*$(o_j) =$ *degré*(o_i) *dans* G_C **do**
      $w\_deg_j \leftarrow \sum_{o_h \in V_G(o_j)} w(e_{jh})$
      if $w\_deg_j > w\_deg_{max}$ **then**
        $w\_deg_{max} = w\_deg_j$
        $i = j$
    ▷ o_i `sommet ayant le plus haut degré pondéré`
  Ajouter o_i dans L_C
  Supprimer sommet o_i et toutes les arêtes adjacentes dans G_C
  forall *sommet* o_j *dont degré* $= 0$ **do**
    Supprimer sommet o_j dans G_C

Grâce au graphe de conflits G_C, nous avons accès au nombre de conflits impliquant chaque individu, grâce au degré des sommets du graphe. À partir de ce graphe G_C, pour savoir quels individus déplacer pour modifier le clustering, nous parcourons les sommets du graphe, chacun représentant un individu, par degré décroissant. Dans cet ordre, les sommets sont ajoutés à la liste L_C. Lorsqu'un sommet est ajouté à la liste, il est supprimé dans G_C, ainsi que toutes les les contraintes de conflit impliquant ce conteneur (soit les arêtes adjacentes à son sommet dans G_C), afin de ne pas traiter les deux conteneurs d'une paire en conflit. À tout moment, si un sommet devient de degré nul, il est alors supprimé de G_C.

Des cas particuliers doivent cependant être pris en compte lors du parcours des sommets du graphe, pour choisir l'individu à ajouter à la liste L_C. En effet, si plusieurs sommets ont le même degré (supérieur à 1) dans G_C, alors nous choisissons celui ayant le plus

haut degré pondéré, calculé comme la somme des valeurs des variables duales ayant levé un conflit impliquant cet individu. De plus, si certains conteneurs n'apparaissent qu'une seule fois dans les contraintes de conflit (le degré du sommet est donc égal à 1), nous avons choisi de traiter en priorité le plus éloigné du profil moyen du cluster auquel il est assigné dans C_{t-1}.

Enfin, nous décrivons en Algorithme 8 comment modifier C_{t-1} pour obtenir un nouveau clustering C_t en utilisant la liste L_C.

Algorithme 8 : Modification du clustering

Input : Clustering C_{t-1}, Données D_t, Liste L_C
Output : Clustering C_t, $\overline{u_{ijt}}$

```
1  C_t ← C_{t-1}
2  forall Individu o_i de L_C do
3      d_{min_i} = +inf
4      forall Cluster c_k dans C_{t-1} do          ▷ a quel cluster affecter o_i ?
5          Calculer d_{i,k} distance entre profil moyen c_k et o_i sur D_t
6          if d_{i,k} < d_{min_i} then
7              d_{min_i} = d_{i,k}
8              new_c_i = c_k
9      Assigner o_i au cluster new_c_i dans C_t
10     Mettre à jour profil moyen new_c_i avec o_i
11 forall (o_i, o_j) assignés au même cluster dans C_t do
12     Ajouter contrainte mustLink_C dans CP
```

Pour chaque conteneur, pris dans l'ordre d'insertion dans la liste L_C (comme une file), nous calculons sa distance avec les profils moyens des autres clusters, et l'attribuons au cluster dont il est le plus proche. Le cluster auquel il était assigné dans C_{t-1} n'est pas exclu, ce qui implique que l'individu peut être réassigné au même cluster que précédemment. Dès qu'un individu est assigné à un cluster, nous recalculons le profil moyen du cluster auquel il est assigné en y ajoutant la consommation du conteneur correspondant.

Une fois le nouveau clustering C_t obtenu, nous modifions en conséquence les variables $u_{i,j}$, ajoutons les nouvelles contraintes Equation 3.21 dans $\mathcal{CP}$, et résolvons le problème $\mathcal{CCP}$ associé, afin de récupérer les nouvelles valeurs de variables duales associées aux contraintes $mustLink_C$ Equation 3.21. Celles-ci pourront donc être comparées pour l'itération suivante.

3.4.2 Mise à jour du clustering - Exemple

Pour illustrer cette étape de mise à jour de clustering, nous utilisons le même jeu de données d'exemple utilisé en Section 3.3, avec une fenêtre de temps $\tau = 3$. La fenêtre

de temps sur laquelle est exploitée la première itération de la boucle comprend donc les points de données $\{4; 5; 6\}$, illustrés en Figure 3.8. Le profil moyen des clusters résultant de l'itération précédente (de la phase d'initialisation) sur cette fenêtre de temps est illustré en Figure 3.9.

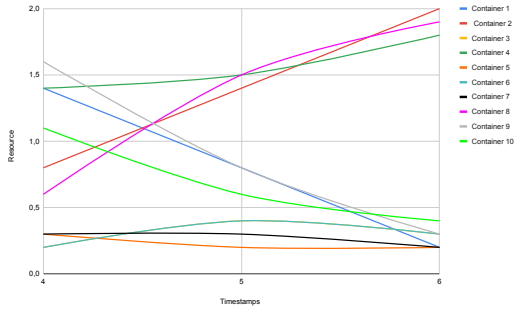


FIGURE 3.8 – Fenêtre de temps utilisée pour la première itération de boucle

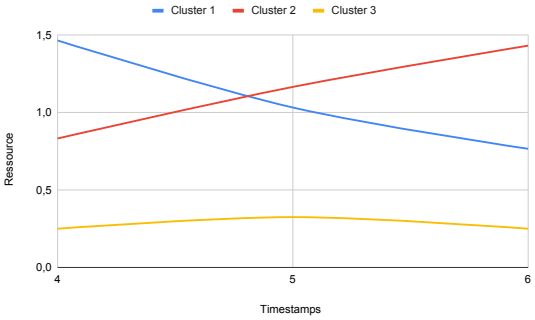


FIGURE 3.9 – Profils moyens des clusters avec C_0

La comparaison des valeurs des variables duales obtenues entre la fenêtre actuelle et la précédente (la phase d'initialisation) donne le graphe de conflit représenté en Figure 3.10.

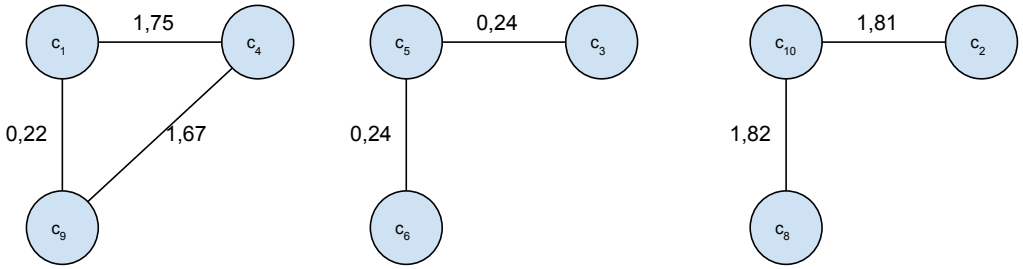


FIGURE 3.10 – Graphe de conflits du clustering G_C sur la fenêtre $\{4; 5; 6\}$

Afin de construire la liste L_C des individus à réassigner pour modifier le clustering, nous parcourons les sommets par degré décroissant, et par degré pondéré décroissant pour les sommets de même degré. Nous pouvons voir qu'il y a 5 sommets dont le degré est égal à 2. Ces sommets sont les suivants, ordonnés par degré pondéré décroissant : $[o_{10}(3.63); o_4(3.42); o_1(1.97); o_9(1.89);$
$o_5(0.48)]$. Nous décrivons ci-dessous les étapes pour construire la liste des individus, en veillant bien à mettre à jour le graphe G_C au fur et à mesure (dont l'évolution est illustré en Figure 3.11) :

1. L'individu o_{10} est ajoutée à la liste : son sommet ainsi que les arêtes adjacentes à son sommet dans G_C sont supprimées. Les sommets o_2 et o_8 étant maintenant de degré 0, ils sont également supprimés.
2. La liste des sommets de degré 2 dans G_C est maintenant la suivante : $[o_4(3.42); o_1(1.97); o_9(1.89); o_5(0.48)]$. L'individu o_4 est donc ajouté à la liste : son sommet ainsi

que les arêtes adjacentes à son sommet dans G_C sont supprimées. Les sommets o_1 et o_9 sont maintenant de degré 1.

3. Le seul sommet de degré 2 dans G_C est le sommet o_5 : il est donc ajouté à la liste, supprimé dans G_C, tout comme les arêtes qui lui sont adjacentes. Les sommets o_3 et o_6 étant maintenant de degré 0 sont supprimés de G_C.
4. Les seuls sommets restants sont les sommets o_1 et o_9. Leur distance au profil moyen de leur cluster dans C_{t-1} (soit c_1) est donc calculée : $d_{1,1} = 1.93$; $d_{9,1} = 1.91 \rightarrow$ le sommet o_1 est ajouté à la liste et supprimé de G_C, tout comme o_9.
5. Le graphe G_C est maintenant vide. La liste des individus à réassigner est donc finalement $[o_{10}; o_4; o_5; o_1]$.

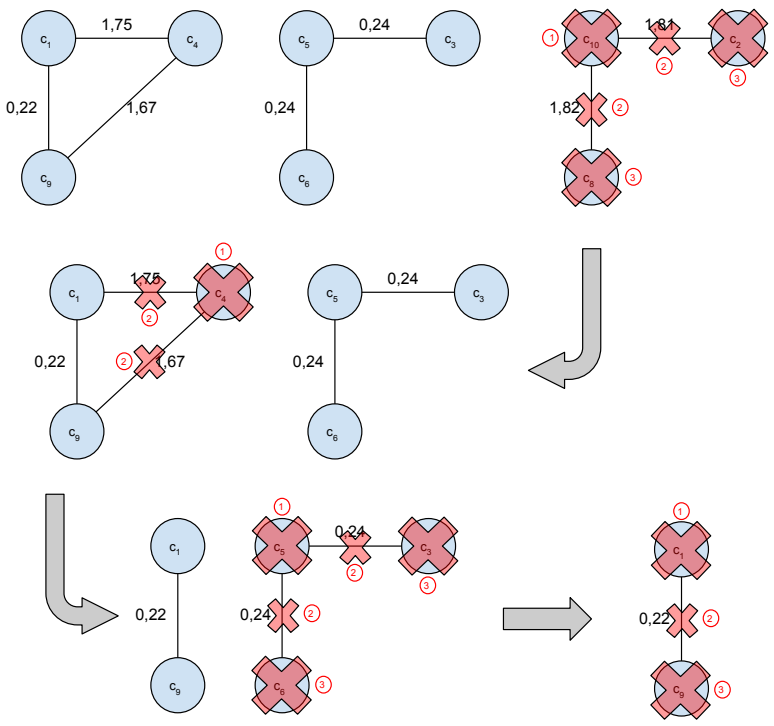


FIGURE 3.11 – Evolution du graphe de conflits du clustering G_C en construisant la liste L_C

Une fois la liste des individus à réassigner obtenue, nous recalculons les profils moyens des clusters, sans prendre en compte les individus de la liste. Ces nouveaux profils moyens sont visibles en Figure 3.12. Nous parcourons ensuite la liste des individus pour les assigner de nouveau au cluster le plus proche, en considérant la consommation du conteneur et le profil moyen du cluster :

1. L'individu o_{10} est considéré en premier : $d_{10,1} = 0.55$; $d_{10,2} = 0.91$; $d_{10,3} = 1.81 \rightarrow$ l'individu o_{10} est donc assigné au cluster c_1. Le profil moyen de ce dernier est mis à jour en y ajoutant la consommation de o_{10}.
2. De la même manière, l'individu o_4 est assigné au cluster c_2, dont le profil moyen est mis à jour avec la consommation de o_4.

3. Les individus o_5 et o_1 sont réassignés respectivement aux clusters c_3 et c_1, qui étaient les clusters auxquels ils étaient assignés dans C_0.

Une fois l'ensemble des individus de la liste réassignés, nous avons notre nouveau clustering C_1, résultant de cette première itération de boucle. Les nouveaux profils moyens des clusters sur cette même fenêtre de temps sont visibles en Figure 3.13.

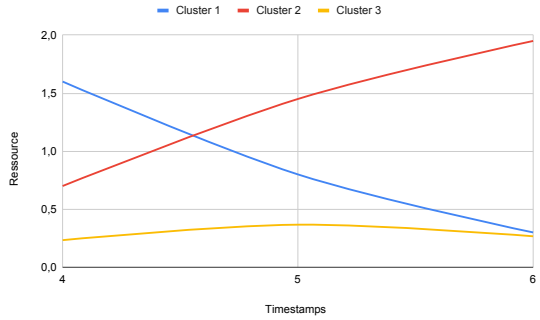


FIGURE 3.12 – Profils moyens des clusters avec C_0 sans considérer les individus à réassigner

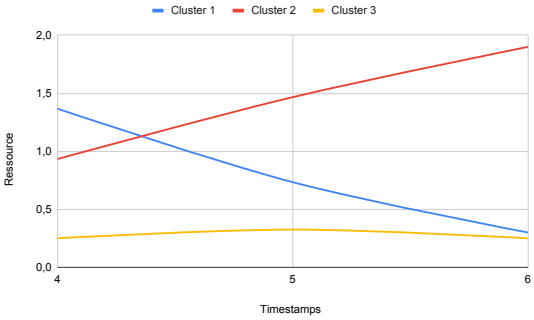


FIGURE 3.13 – Profils moyens des clusters avec C_1

3.4.3 Mise à jour du placement $A_{t-1} \rightarrow A_t$

Avant de détailler la méthode de mise à jour du placement, nous rappelons les notations utilisées dans cette partie en Tableau 3.5.

Notation	Signification
I	Nombre d'individus (conteneurs)
M	Nombre de nœuds
$\{o_1, o_2, \ldots, o_I\}$	Ensemble des individus
$\{n_1, n_2, \ldots, n_M\}$	Ensemble des nœuds
C_t	Solution de clustering au temps t
A_t	Solution de placement au temps t
D_t	Données concernant le temps t (entre $t-\tau$ et t)
$\mathcal{AP}$	Modélisation de notre problème de placement
$\mathcal{CAP}$	Relaxation continue du problème $\mathcal{CAP}$
$mustLink_A$	Contraintes ajoutées au problème $\mathcal{AP}$ pour fixer la solution actuelle
$\overline{v_{ijt}}$	Variables duales associées aux contraintes $mustLink_A$ au temps t
ϵ_A	Seuil de tolérance pour comparer les valeurs des variables $\overline{v_{ijt}}$
G_A	Graphe de conflit de la solution de placement actuelle

TABLEAU 3.5 – Index des notations utilisées pour le placement.

Une fois que le clustering a été mis à jour en fonction des nouvelles données D_t, nous utilisons son résultat pour évaluer le placement actuel des conteneurs sur les différents nœuds de l'infrastructure. Tout comme pour le clustering, nous commençons par lever les conflits parmi les conteneurs placés sur le même nœud dans le placement A_{t-1}, et

ainsi déplacer ces individus pour obtenir un nouveau placement A_t. La première étape de découverte des conflits est détaillée en Algorithme 9.

Algorithme 9 : Construction du graphe de conflit G_A pour le placement

Input : Clustering C_t, Placement A_{t-1}, $\overline{v_{ij(t-1)}}$, ϵ_A
Output : Graphe G_A

```
Construire problème 𝒜𝒫 avec clustering C_t
Ajouter contraintes mustLink_A dans 𝒜𝒫 via solution A_{t-1}
Résoudre 𝒞𝒜𝒫 relaxation continue de 𝒜𝒫
Récupérer valeurs des variables duales v_ijt associées aux contraintes mustLink_A
if v_ijt > v_ij(t-1)(1 + ε_A) then
    Ajouter couple (o_i, o_j) dans G_A
```

De la même manière que pour le clustering, l'évaluation de la solution de placement A_{t-1} est réalisée grâce au problème d'optimisation $\mathcal{AP}$. Comme détaillé en Section 3.2, les variables $u_{i,j}$ mises à jour dans l'étape précédente sont ici considérées comme des données et utilisées afin de décrire l'objectif de placement. La solution de placement A_{t-1} est elle renseignée dans $\mathcal{AP}$ grâce à l'ajout de contraintes $mustLink_A$ qui, pour tout couple d'individus (o_i, o_j) placés sur le même nœud dans A_{t-1} fixe la variable correspondante $v_{i,j}$ à 1 :

$$mustLink_A : v_{i,j} = 1 \ \forall (i,j) \text{ placés sur le même nœud dans } A_{t-1} \tag{3.24}$$

Le problème d'optimisation $\mathcal{AP}$ modélisant le placement, détaillé en Section 3.2, est rappelé en Figure 3.14 en y ajoutant les contraintes $mustLink_A$ décrites ci-dessus.

$$\min \quad \sum_{(i,j)\in[1..I]} u_{i,j}v_{i,j} + (1-u_{i,j})v_{i,j}d_{i,j} \tag{3.25a}$$

$$cons_{m,t} \leq cap_m \qquad \forall m \in [1..M],\ t \in [1..T] \tag{3.25b}$$

$$\sum_{m=1}^{M} x_{i,m} = 1 \qquad \forall i \in [1..I] \tag{3.25c}$$

$$x_{i,m} \leq a_m \qquad \forall i \in [1..I],\ m \in [1..M] \tag{3.25d}$$

$$\sum_{m=1}^{M} a_m \leq max\_nodes \tag{3.25e}$$

$$v_{i,j} = \sum_{m=1}^{M} x_{i,m}x_{j,m} \qquad \forall (i,j) \in [1..I] \tag{3.25f}$$

$$v_{i,j} = 1 \qquad \forall\ (o_i, o_j) \in meme\ n_m\ dans\ A_{t-1} \tag{3.25g}$$

FIGURE 3.14 – Problème de placement $\mathcal{AP}$ avec les contraintes $mustLink_A$.

Tout comme pour le problème $\mathcal{CP}$ pour le clustering, ici les contraintes 3.25f sont quadratiques. Pour rendre le problème $\mathcal{AP}$ linéaire, la même méthode de linéarisation

est appliquée, en remplaçant cette contrainte par l'ensemble des contraintes suivantes :

$$z_{i,j,m} \leq x_{i,m} \qquad \forall i \in [1..I], m \in [1..M] \tag{3.26a}$$

$$z_{i,j,m} \leq x_{j,m} \qquad \forall j \in [1..I], m \in [1..M] \tag{3.26b}$$

$$z_{i,j,m} \geq x_{i,m} + x_{j,m} - 1 \qquad \forall (i,j) \in [1..I], m \in [1..M] \tag{3.26c}$$

$$z_{i,j,m} \geq 0 \tag{3.26d}$$

Nous résolvons ensuite la relaxation continue de $\mathcal{AP}$, appelée $\mathcal{CAP}$, afin de récupérer les valeurs des variables dualées associées aux contraintes $mustLink_A$, appelées $\overline{v_{ij}}$. Afin de considérer l'augmentation de ces valeurs d'une itération à une autre, nous considérons un seuil de tolérance spécifique au placement appelé ϵ_A, ainsi que $\overline{v_{ij(t-1)}}$ la valeur de la variable duale $\overline{v_{ij}}$ à l'itération précédente, et $\overline{v_{ijt}}$ sa valeur actuelle. Pour chaque contrainte $mustLink_A$ définie par l'Équation 3.24, si $\overline{v_{ijt}} > \overline{v_{ij(t-1)}}(1 + \epsilon_A)$, alors on considère les conteneurs o_i et o_j en conflit.

Le graphe de conflit G_A obtenu à la fin de l'étape précédente nous donne l'ensemble des paires de conteneurs actuellement colocalisés mais dont la séparation permettrait d'obtenir une meilleure solution, en considérant la superposition de leur profil. La gestion de ces conflits, dans le but d'obtenir la liste des conteneurs à déplacer, est détaillée en Algorithme 10.

La construction de la liste L_A des conteneurs à déplacer pour obtenir un nouveau placement A_t à partir de A_{t-1} est similaire à la construction de la liste L_C pour le clustering : nous parcourons les sommets du graphe de conflit G_A par ordre de degré décroissant, et ajoutons les sommets successivement dans la liste, en mettant à jour le graphe G_A à chaque ajout dans L_A (supprimer de G_A les sommets ajoutés dans L_A, les arêtes adjacentes à ces sommets, et les sommets de degré nul). La gestion des sommets de même degré est identique que lors de la création de L_C pour le clustering : parmi plusieurs sommets de même degré, nous choisissons le sommet ayant le plus haut degré pondéré, calculé comme la somme des valeurs des variables duales ayant levé un conflit impliquant cet individu. Concernant les sommets de degré 1, pour chaque paire de conteneurs nous calculons la variance de consommation du nœud sur lequel ces conteneurs sont placés dans A_{t-1}, et retenons le conteneur dont la suppression donne la variance la plus faible sur le nœud.

Une fois le graphe G_A vide et la liste L_A, il reste à replacer les conteneurs de cette dernière pour obtenir A_t. Ce replacement est détaillé en Algorithme 11.

Pour chaque conteneur de la liste L_A, pris par consommation maximale décroissante sur la période actuelle, nous attribuons le conteneur au nœud pour lequel la variance de consommation totale, en y ajoutant la consommation du conteneur considéré sur les données D_t, est la plus faible.

Algorithme 10 : Construction de la liste de réassignation L_A

Input : Clustering C_t, Placement A_{t-1}, Données D_t, Graphe G_A
Output : Liste L_A

$L_A \leftarrow [\,]$
Trier les sommets du graphe G_A par degré décroissant
forall *Sommet* o_i *de* G_A *par degré décroissant* **do**
  if *degré*$(o_i) = 1$ **then** ▷ `cas où degré = 1`
    $o_j \leftarrow$ sommet voisin de o_i dans G_A
    $(new\_var_{i,m}, new\_var_{j,m}) \leftarrow$ variance de consommation de n_m auquel o_i et o_j sont assignés dans A_{t-1}, en enlevant respectivement la consommation de o_i et o_j
    if $new\_var_i > new\_var_j$ **then**
      $i = j$
    ▷ o_i `individu créant le plus d'instabilités de consommation sur` n_m
  else if $\exists\ o_j$ *t.q.* *degré*$(o_j) =$ *degré*(o_i) **then** ▷ `cas où plusieurs sommets de même degré`
    $w\_deg_{max} = 0$
    forall *Sommet* o_j *t.q.* *degré*$(o_j) =$ *degré*(o_i) *dans* G_A **do**
      $w\_deg_j \leftarrow \sum_{o_h \in V_G(o_j)} w(e_{jh})$
      if $w\_deg_j > w\_deg_{max}$ **then**
        $w\_deg_{max} = w\_deg_j$
        $i = j$
    ▷ o_i `sommet ayant le plus haut degré pondéré`
  Ajouter o_i dans L_A
  Supprimer sommet o_i et toutes les arêtes adjacentes dans G_A
  forall *sommet* o_j *dont degré* $= 0$ **do**
    Supprimer sommet o_j dans G_A

Tout comme pour le clustering, les variables $v_{i,j}$ sont mises à jour grâce à cette nouvelle solution de placement A_t, tout comme les contraintes $mustLink_A$ dans le problème $\mathcal{AP}$. La relaxation continue $\mathcal{CAP}$ de ce dernier est résolue afin de mettre à jour les valeurs des variables duales $\overline{v_{ijt}}$ avant de pouvoir être utilisées dans l'itération de boucle suivante.

Algorithme 11 : Mise à jour du placement

Input : Clustering C_t, Placement A_{t-1}, Données D_t, Liste L_A
Output : Placement A_t, $\overline{v_{ijt}}$

```
A_t ← A_{t-1}
Trier les individus o_i de L_A par max_{D_t}(conso_{o_i,t}) décroissant
forall Individu o_i de L_A do
    var_{min_i} = +inf
    forall nœuds n_m dans A_{t-1} do
        Calculer var_{i,m} variance de consommation du nœud n_m en y ajoutant o_i
        if var_{i,m} < var_{min_i} then
            var_{min_i} = var_{i,m}
            new_n_i = n_m
    Assigner o_i au nœud new_n_i dans A_t
forall (o_i, o_j) colocalisés dans A_t do
    Ajouter contrainte mustLink_A dans AP
```

3.4.4 Mise à jour du placement - Exemple

Pour illustrer cette étape de mise à jour du placement, nous reprenons la suite de l'exemple présent en Section 3.4.2. Sur la même fenêtre de temps, comprenant les points de données $\{4; 5; 6\}$, l'évolution de consommation de ressource des 2 nœuds considérés suite au résultat de placement A_0 est visible en Figure 3.16.

La comparaison des valeurs des variables duales obtenues entre la fenêtre actuelle et la précédente (la phase d'initialisation) donne le graphe de conflit représenté en Figure 3.15.

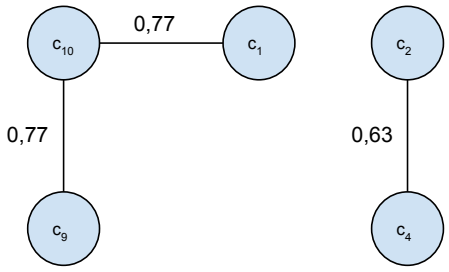


FIGURE 3.15 – Graphe de conflits du placement G_A sur la fenêtre $\{4; 5; 6\}$

À partir de ce graphe de conflits G_A nous construisons la liste L_A des conteneurs à déplacer pour obtenir un nouveau placement :

1. Le sommet o_{10} est le seul de degré 2 : il est donc ajouté à L_A. Ses arêtes adjacentes sont supprimées dans G_A, ainsi que les sommets o_1 et o_9 devenant de degré nul.
2. Les sommets o_2 et o_4 sont les derniers à considérer : nous calculons la variance de consommation sur la fenêtre actuelle du nœud n_2 (sur lequel ils sont placés dans A_0) en supprimant l'un puis l'autre. La différence de consommation de n_2 sans o_2

puis sans o_4 est visible en Figure 3.17. La variance de consommation de n_2 sans o_2 est 0.01, celle sans o_4 est 0.17 → nous ajoutons o_2 dans L_A puis supprimons o_4 de G_A.

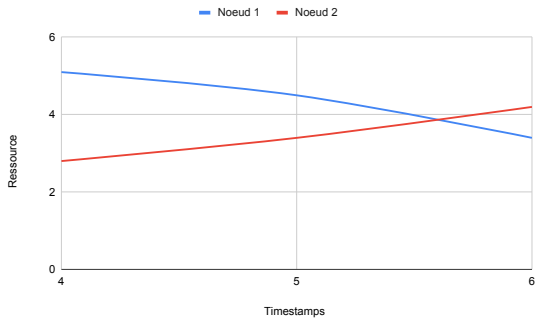


FIGURE 3.16 – Evolution de consommation sur les nœuds durant la période $\{4; 5; 6\}$

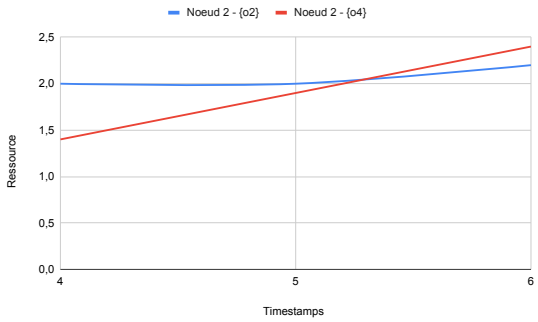


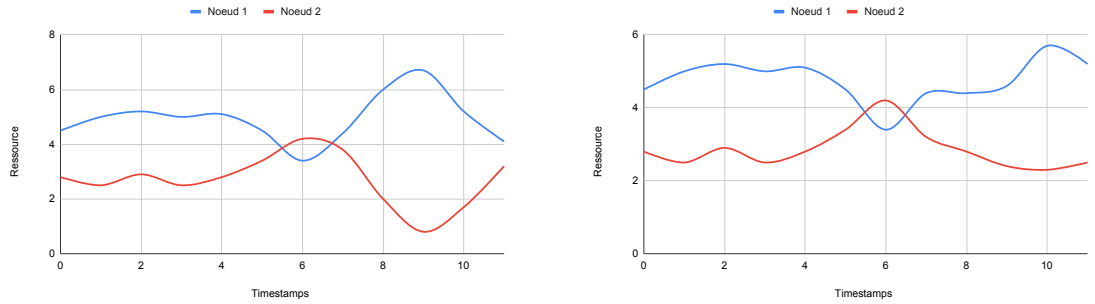
FIGURE 3.17 – Consommation de ressource du nœud n_2 sans o_2 et sans o_4

Les consommations de ressource des nœuds n_1 et n_2 pour la période actuelle sont recalculées en supprimant les données de consommation des conteneurs apparaissant dans la liste L_A (soit respectivement o_{10} et o_2). Cette dernière liste peut être parcourue pour replacer les conteneurs, en les assignant au nœud résultant de la plus faible variance de consommation :

1. L'individu o_2 est considéré en premier, ayant un pic maximum de consommation supérieur à o_{10} : $var_{2,1} = 0.042$; $var_{2,2} = 0.33$ → le conteneur o_2 est placé sur le nœud n_1, dont la consommation totale est mise à jour avec la consommation de o_2.
2. Le conteneur o_{10} reste à placer : $var_{10,1} = 0.56$; $var_{10,2} = 0.55$ → o_{10} est placé sur le nœud n_2.

Une fois l'ensemble des individus de la liste réassignés, nous avons notre nouveau placement A_1, résultant de cette première itération de boucle. Il est à noter que les consommations de ressource recalculées au cours de la mise à jour du placement sont des simulations pour aider à la décision, en réalité le changement de placement ne prend effet qu'après la fenêtre de temps actuelle.

La Figure 3.18 montre l'évolution de consommations de ressource des nœuds sur l'ensemble du jeu de données d'exemple en appliquant la solution de la phase d'initialisation A_0 sans la modifier au fil du temps, comparée à la même évolution en modifiant la solution A_0 vers A_1 après la première itération de la boucle. En plus d'un gain de stabilité, nous remarquons que la modification du placement a permis d'éviter un dépassement de capacité sur le nœud n_1.



(a) Placement de la phase d'initialisation sans changement

(b) Placement mis à jour par la boucle

FIGURE 3.18 – Comparaison des évolutions de consommations sur les nœuds en utilisant seulement la solution de la phase d'initialisation (a) et en appliquant la solution en première itération de boucle (b)

3.5 Déclenchement d'une alerte et réparation locale

Lors du parcours des points de données pendant la période T_{run}, entre l'exécution de deux boucles successives, il se peut que la consommation totale d'un nœud dépasse un certain seuil maximal, nommé $node_{max}$. Par défaut, ce seuil est fixé à 95% de la capacité maximale d'un nœud. Si ce seuil est atteint, une alerte est levée afin de modifier, en mode "asynchrone", le placement actuel pour respecter ce seuil (et ainsi éviter des dépassements de capacité, occasionnant des incidents de production). Cette réparation est locale et développée de manière à trouver une nouvelle solution de placement le plus rapidement possible. Lors de cette réparation, nous considérons les données de consommation de la fenêtre D_t comprenant les données entre les temps $t - \tau$ et t, avec t étant le moment où l'alerte est levée. Cette méthode de réparation est détaillée en Algorithme 12.

Cette réparation se fait de manière locale, de manière à corriger l'erreur le plus rapidement possible, afin de ne rester dans cet état d'alerte le moins de temps possible. En effet, cet état représente une situation critique au niveau de l'environnement de production, pouvant entraîner des incidents ou des arrêts de service. L'apparition de ce type d'alerte est d'ailleurs considéré comme un critère à minimiser (voir Section 4.2).

3.6 Conclusion / Discussion

Afin d'obtenir des solutions de placement des conteneurs adaptées à l'évolution de consommation de ces conteneurs dans le temps, nous élaborons une stratégie de placement en utilisant les profils de consommation de ces individus.

Le profilage des conteneurs est le résultat d'une étape de clustering. Ce problème de clustering est modélisé par un PLNE, dont certaines variables de décision sont utilisées ensuite, en les considérant comme des données, dans la modélisation du problème de

Algorithme 12 : Réparation du placement suite à une alerte

Input : Données D_t, Placement A_{t-1}, Nœud n_m ayant levé l'alerte
Output : Placement A_t

```
1  while conso_{m,t} >= node_max do
2      o_i ← conteneur aléatoire placé sur n_m dans A_{t-1}
3      done ← False
4      l ← m + 1
5      while not done do
6          if l = m then                    ▷ o_i n'a pu aller sur aucun nœud : nœud
             supplémentaire requis
7              M++
8              l = M
9              Placer o_i sur n_l
10         if conso_{l,t} + conso_{o_i,t} < node_max then
11             Placer o_i sur n_l
12         else
13             l ← (l + 1)%M
```

placement par un PLNE également.

Afin de répondre à la dynamicité de nos problèmes de clustering et de placement, nous incorporons dans ces modèles mathématiques l'information des solutions (respectivement de clustering et de placement) courantes à travers des contraintes (respectivement $mustLink_C$ et $mustLink_A$) permettant de fixer les variables de décision liant les conteneurs entre eux (respectivement dans un même cluster et dans un même nœud). L'évolution des valeurs des variables duales associées à ces contraintes nous permettent de remettre en cause les solutions existantes, en construisant un graphe de conflit pour chacun de nos problèmes (clustering et placement). Ces graphes de conflit, à travers la considération des degrés de leurs sommets, nous permettent d'obtenir des listes de conteneurs qui sont susceptibles de changer respectivement de cluster et de nœud, en ayant une vision globale des conflits levés. Ces changements sont réalisés par des heuristiques de replacement, vérifiant l'amélioration effective de la solution en cas de changement.

Ces mécanismes sont implémentés à travers une boucle, s'exécutant en continu à l'arrivée de nouvelles données. Cette boucle suit un fonctionnement à deux niveaux :

— la mise à jour du clustering en fonction des nouvelles données de consommation ;

— l'intégration des changements de clustering dans la modélisation du placement, permettant à son tour sa mise à jour.

Les solutions initiales du clustering et du placement sont obtenues par des heuristiques :

— le clustering est calculé par l'algorithme *K-means*, dont l'aspect itératif est adapté à la dynamicité de notre contexte ;

— le placement est obtenu grâce au développement d'une heuristique originale, utilisant le résultat du *K-means* pour colocaliser les conteneurs en fonction du cluster auquel ils appartiennent, en considérant la variance de la somme des profils moyens de ces clusters.

La construction des graphes de conflits se fait via la résolution de programmes linéaires, dont la construction ne se fait qu'une seule fois (mise à jour des données du programme à chaque itération de la boucle) et dont la résolution successive d'une itération à une autre permet l'utilisation de la précédente résolution pour accélérer la résolution du problème actuel (voir Section 4.1). Les heuristiques de modifications des solutions de clustering puis de placement à partir des listes de réassignation (respectivement L_C et L_A) ont une complexité respectivement en $O(|L_C|K)$ et en $O(|L_A|M)$. Ces étapes permettent la modification rapide des solutions, dont les temps de calcul sont détaillés en Section 4.4.

Les profils de consommation, à travers le résultat du clustering lui-même mis à jour grâce à des méthodes d'optimisation, guide donc en continue le placement des conteneurs sur une infrastructure.

L'efficacité et l'utilité de cette méthode, à travers la définition de différents critères et indicateurs, ainsi que son implémentation concrète à travers une librairie, sont discutées dans le chapitre suivant.

Chapitre 4

Implémentation et expérimentations

Sommaire

Dans ce troisième chapitre nous présentons les moyens mis en œuvre pour tester l'efficacité de la méthode décrite précédemment.

Dans cette optique, nous avons développé une librairie Python dont l'acronyme est COTS (pour *Continuous Optimization for Time Series*), dont la description complète sera l'objet de la première section ci-après.

La deuxième partie de ce chapitre est quant à elle consacrée à l'évaluation de la méthode, par le biais d'expérimentations menées avec la librairie COTS, dont nous discutons les résultats en dernière partie.

4.1 Présentation générale de la librairie COTS

Afin d'implémenter notre méthode présentée dans le Chapitre précédent, nous avons développé dans le cadre de cette thèse une librairie *Python* nommée COTS.

Cette librairie permet à l'utilisateur d'obtenir une ou plusieurs stratégies de configuration pour un jeu de données, définissant dans notre cas d'usage un placement à partir de l'évolution des consommations de ressources représentées par ce jeu de données.

Nous décrivons dans un premier temps le fonctionnement de la libraire, avant d'évoquer son développement, et en détaillant son architecture modulaire.

4.1.1 Fonctionnement de COTS

L'utilisation de la librairie COTS se fait via la commande suivante dans un terminal :

```
cots path --k --tau --method --cluster_method --param --
    output --tolclust --tolplace
```

Parmi les arguments listés, seul *path* est obligatoire. Ci-dessous sont décrits les détails de ces arguments :

- *path* : chemin d'accès au répertoire contenant les fichiers de données
- k : nombre de clusters utilisés pour le clustering
- *tau* : taille de la fenêtre de temps utilisée pour exécuter les boucles
- *method* : choix de la méthodologie globale
- *cluster_method* : choix de la méthode pour mettre à jour le clustering
- *param* : utilise un fichier de paramètres spécifique
- *output* : utilise un répertoire spécifique pour les fichiers de résultats
- *tolclust* : seuil utilisé pour comparer les variables duales liées au clustering (ϵ_C)
- *tolplace* : seuil utilisé pour comparer les variables duales liées au placement (ϵ_A)

Dans le répertoire fourni contenant les données (via l'argument *path* obligatoire), représentées par des fichiers *.CSV*, la librairie doit y trouver au minimum deux fichiers, et deux autre fichiers optionnels, tels que :

- Un fichier *container_usage.csv* obligatoire : contient les traces de consommations des conteneurs ;
- Un fichier *node_meta.csv* obligatoire : contient les informations, notamment les capacités de ressources, concernant les nœuds ;
- *Optionnel* Un fichier *node_usage.csv* : contient les traces de consommations des nœuds. S'il n'est pas fourni, il est construit à partir des traces de consommations

des conteneurs et du placement initial du dataset. Ce fichier optionnel peut être utile dans le cas où les nœuds sont également utilisés par d'autres éléments (applications, processus) non accessibles par le jeu de données ;

— *Optionnel* Un fichier *params.json* : contient les différents paramètres utilisés par la librairie. S'il n'est pas fourni, le fichier de paramètres par défaut, fourni dans la librairie, est utilisé. L'ensemble des paramètres pouvant être renseignés dans ce fichier sont décrits en Annexe A.1.

Le fonctionnement global de la librairie COTS, et l'implémentation de la boucle, sont résumés dans le schéma Figure 4.1.

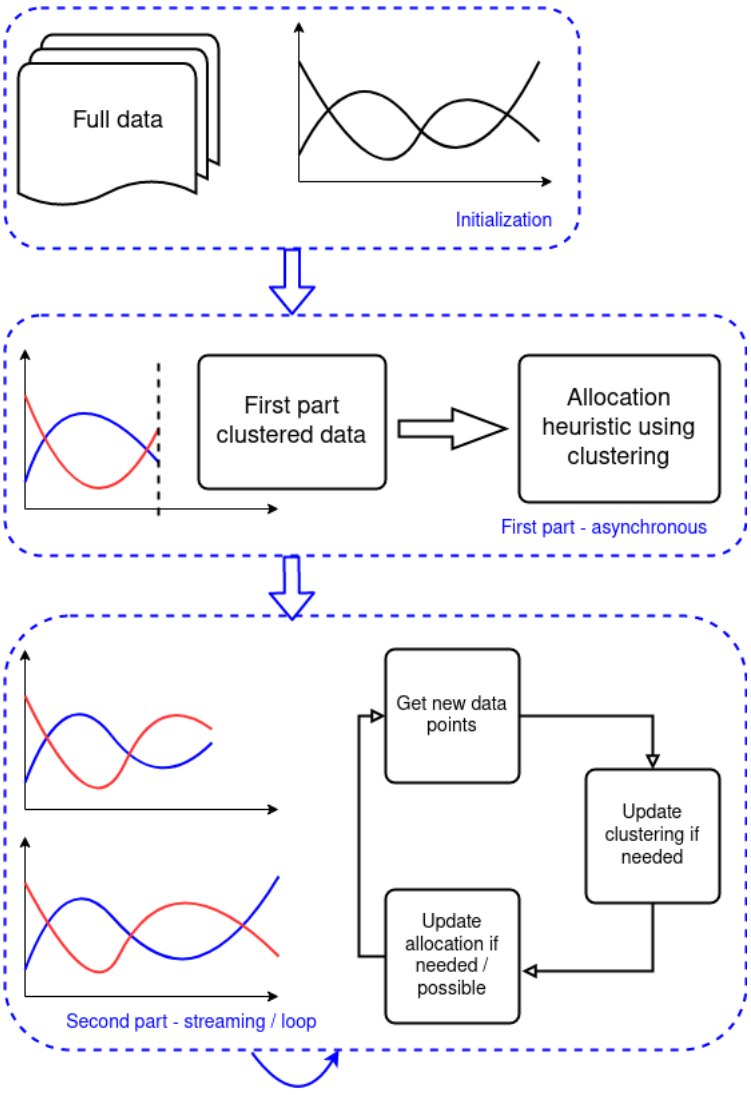


FIGURE 4.1 – Schéma de fonctionnement global de COTS

Au lancement de COTS, les données de consommations sont divisées en deux parties, dépendamment du paramètre *sep_time* : les données concernant les consommations pour des temps inférieurs à *sep_time* formeront la partie des données d'analyse pour la période d'observation T_{obs}, dont le but est d'obtenir les premières solutions de clustering et de placement (voir Section 3.3) ; alors que les données concernant les consommations

pour des temps supérieurs à *sep_time* formeront la deuxième partie dans laquelle nous exécutons les boucles (voir Section 3.4).

4.1.2 Architecture générale

La structure globale du code de COTS, schématisée avec tous les fichiers en Figure 4.2 s'articule autour d'un point d'entrée classique (fichier `main.py`).

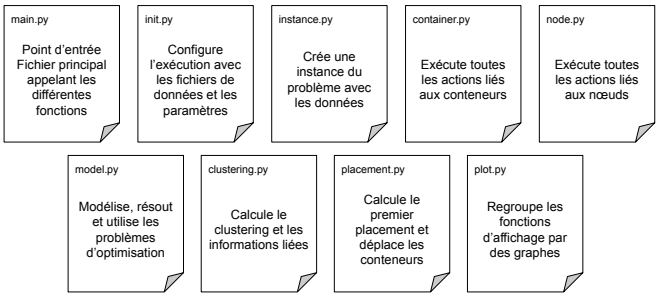


FIGURE 4.2 – Schéma des fichiers constituant *COTS*

Le rôle de chacun de ces fichiers en fonction des actions de la méthode est illustré dans un premier temps dans la Figure 4.3 en ce qui concerne la partie regroupant le *preprocessing* et la période d'analyse T_{obs} (détaillée en Section 3.3), puis dans la Figure 4.4 concernant la phase d'exécution des boucles (dont le fonctionnement est détaillé en Section 3.4).

4.1.2.1 Modularité des composants

L'architecture décrite ci-dessus donne un aperçu global des interactions entre les fichiers constituant la brique essentielle de la librairie COTS. Concernant l'implémentation du code, un travail spécifique a été réalisé dans l'optique de rendre ce code le plus modulaire possible. En effet, notre volonté est de laisser de la liberté d'utilisation dans son utilisation. Cette liberté peut se traduire par de simples alternatives à certaines étapes, comme par exemple l'obtention du premier clustering (pendant la période d'observation T_{obs}), où nous utilisons l'algorithme *K-means* mais ont été implémentés une méthode de clustering hiérarchique et une méthode spectrale. Mais elle se traduit surtout, dans son implémentation et sa structure, par la possibilité d'utiliser son propre cas d'usage. Ce dernier point est illustré par deux possibilités, décrites ci-dessous.

Modularité des systèmes de solvers

Comme expliqué précédemment, toutes les actions (construction, résolution, mise à jour, utilisation des variables duales...) concernant les problèmes d'optimisation sont regroupées dans le fichier `model.py`. Cependant, ce fichier est une interface dans laquelle sont développées les différentes fonctions, en utilisant notamment le module

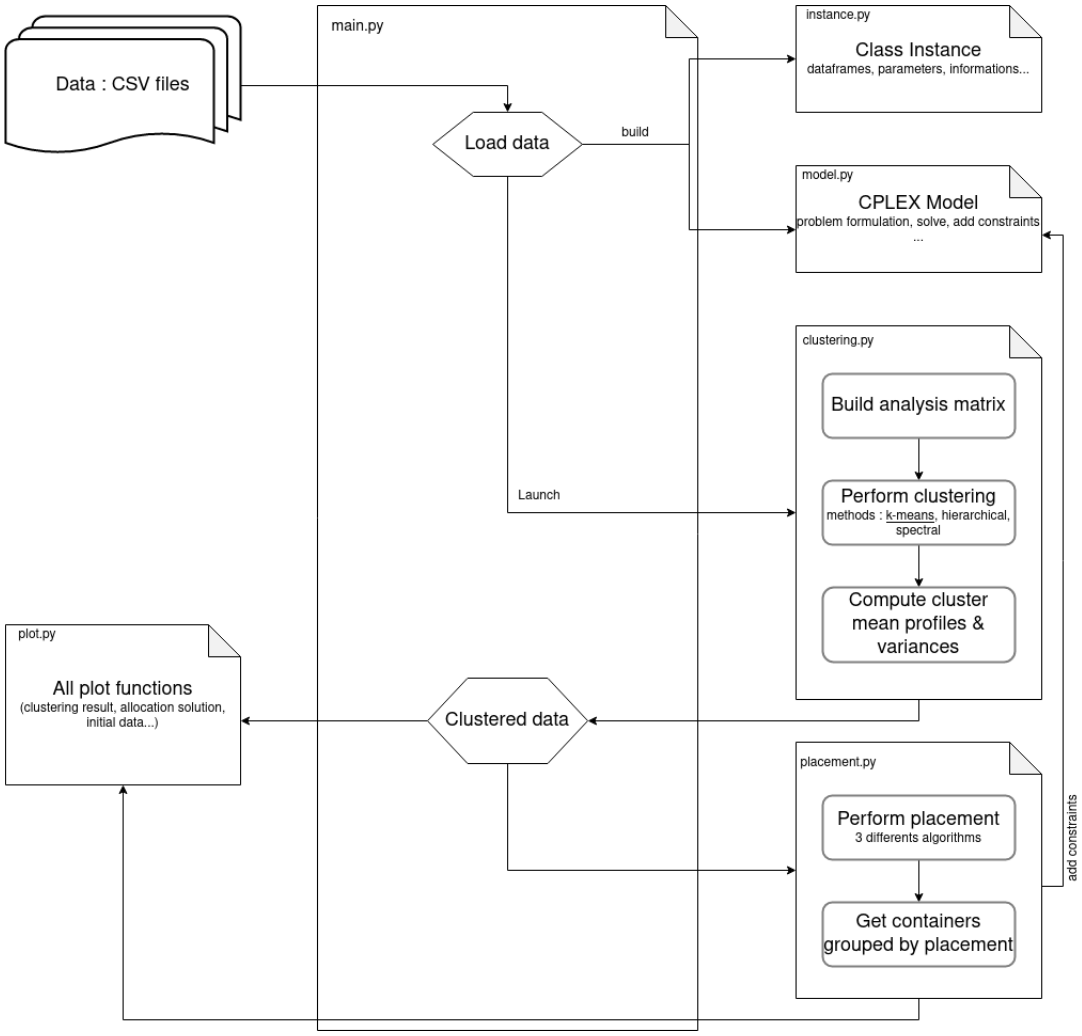


FIGURE 4.3 – Schéma de structure de COTS (période d'analyse)

Python Pyomo [Pyomo, 2020]. En ce qui concerne la modélisation exacte des problèmes, l'utilisateur peut renseigner un fichier précis définissant l'ensemble des contraintes, variables et fonction objectif des problèmes. Par défaut, un fichier définissant les modèles tel que décrits en Section 3.2 est fourni avec la librairie.

De plus, l'interface fait appel à des fonctions haut niveau pour résoudre les problèmes d'optimisation. Ces résolutions font appel à un solver, notamment afin d'obtenir les valeurs des variables duales utilisées dans notre méthode. Il existe plusieurs solvers, dont le plus répandu est IBM ILOG CPLEX Optimization Studio (CPLEX). Ce dernier étant un outil propriétaire, et dans l'optique de reverser notre librairie à la communauté open source, nous laissons la possibilité à l'utilisateur de renseigner le solver qu'il souhaite utiliser, en fournissant lui-même sa propre définition des problèmes et le solver voulu. Lors du développement de COTS, nous avons testé cette modularité avec les solvers CPLEX et GNU Linear Programming Kit (GLPK).

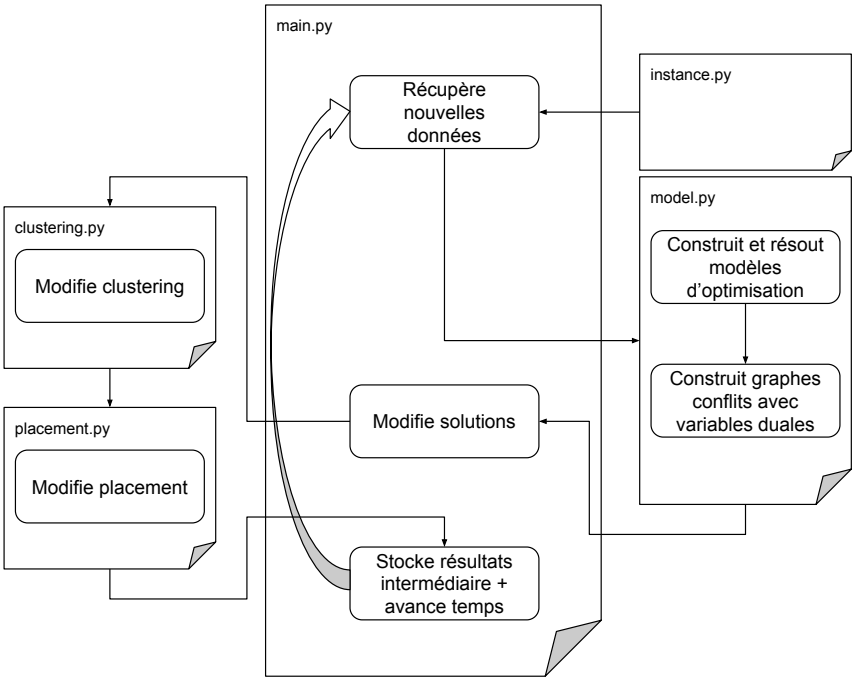


FIGURE 4.4 – Schéma de structure de COTS (période d'exécution des boucles)

Modularité des composants de résolution métier

La librairie COTS a été développée dans le cadre du placement dynamique de conteneurs sur une infrastructure. Son développement répond avant tout au besoin d'implémenter et évaluer la méthode décrite dans le Chapitre précédent. Nous pouvons imaginer que ce type de méthodologie, alliant profilage d'individus et techniques d'optimisation dans un système de boucles, peut être utilisé pour répondre à d'autres problématiques que celui de cette thèse. C'est pourquoi l'aspect métier apparaissant dans la librairie fait également l'objet d'un développement modulaire : toutes les actions concernant le placement des conteneurs (l'heuristique utilisée dans la période d'observation T_{obs}, les déplacements de conteneurs dans les boucles etc ...) sont regroupées dans le fichier `placement.py`. Si l'utilisateur souhaite utiliser la même méthodologie mais l'appliquer à une problématique différente, il devra donc fournir de nouvelles fonctions dans un fichier en lieu et place de `placement.py`.

À titre d'exemple, nous avons exploré au cours de cette thèse le problème du niveau de ressources allouées aux conteneurs. Pour cela, nous avons créé un fichier supplémentaire nommé `allocation.py` pour y développer de premiers cas d'usage et tests, qui sont alors directement utilisés dans la méthodologie principale. Ce travail exploratoire vient étendre la problématique de placement de conteneurs dans une vision plus globale d'une allocation de ressources optimisée, plus de détails sont donnés en Section 5.2.

4.2 Expérimentations

La méthode décrite au chapitre précédent, ainsi que la librairie COTS développé pour implémenter cette dernière, doivent être soumis à une phase d'évaluation, tant pour évaluer l'efficacité globale de la méthode que pour mesurer et justifier l'apport spécifique de certaines étapes de la méthode. Pour cela, nous décrivons point par point l'ensemble des expérimentations réalisées, leur motivation et leur mise en place, tout en décrivant les jeux de données utilisés pour chacun de ces points ainsi que leur provenance.

Notation	Signification
c_1	Critère du nombre de nœuds
c_2	Critère de l'amplitude maximale de consommation sur les nœuds
c_3	Critère de la variance maximale de consommation sur les nœuds
c_4	Critère du nombre d'alertes engendrées

TABLEAU 4.1 – Index des notations pour la description des expérimentations.

4.2.1 Efficacité globale de la méthode

Notre méthode ayant été développée pour répondre à une problématique précise, le placement de conteneurs web, le premier point d'évaluation porte sur l'apport métier de notre méthode. Pour cela, nous identifions un ensemble de critères quantitatifs importants pour le domaine et permettant la comparaison de différentes techniques et / ou de différents scénarios de placement. Ces critères sont les suivants :

— $c_1 = nb_{noeuds} = \sum_{n=1}^{N} a_n$, avec

$$a_n = \begin{cases} 1 & \text{si le nœud } n \text{ est ouvert (au moins 1 conteneur dessus)} \\ 0 & \text{sinon} \end{cases}$$

c_1 représente le nombre de nœuds, que nous cherchons à minimiser.

— $c_2 = ampl_{max} = \max_{\{n \in N\}} (\max_{\{t \in T\}} (conso_{n,t}) - \min_{\{t \in T\}} (conso_{n,t}))$
c_2 représente la plus grande amplitude de consommation parmi tous les nœuds. Ceci indique un niveau de stabilité de consommation sur les nœuds.

— $c_3 = var_{mean} = \frac{1}{N} \sum_{n=1}^{N} \sum_{n=1}^{N} (\overline{cons_n} - cons_{n,t})^2$
c_3 représente la variance moyenne de consommation sur tous les nœuds, qui est un autre indice de stabilité de consommation.

— $c_4 = nb_{alerts}$ est le nombre de fois où la capacité maximale des nœuds a été atteinte. Dans COTS, une fonction de réparation a été implémentée pour toutes les méthodes, déplaçant des conteneurs si un nœud a atteint sa limite (voir Section 3.5). Dans l'environnement d'exécution Docker, qui est la plateforme

d'hypervision ciblée en production , nous voulons éviter les dépassements de capacité (*overloads* [Docker, 2014b]) car ceux-ci peuvent entraîner des comportements critiques, notamment des exceptions, comme c'est le cas avec l'Erreur de mémoire insuffisante (Out Of Memory Exception) (OOME).

Pour pouvoir évaluer la performance de notre méthode vis-à-vis de ces critères, nous avons implémenté 5 stratégies différentes de placement. Elles couvrent à la fois la méthode canonique, telle que présente dans l'hyperviseur Docker, ainsi que des méthodes plus sophistiquées, incluant notre méthode :

— *initial* : Le placement initial, comme fourni dans les jeux de données
— *spread* : Méthode utilisée par défaut dans les scheduleurs [Docker, 2014a], elle a été implémentée pour se comparer à ce qui est traditionnellement utilisé (voir Section 2.1.1). Cette technique consiste à parcourir les conteneurs un par un, et à les assigner sur les nœuds en les parcourant de manière itérative (le premier conteneur sur le premier nœud, le second conteneur sur le second nœud, etc...)
— *iter-consol (IC)* : Cette technique utilise la période d'analyse T_{obs} pour fixer un nombre minimum de nœuds requis (en additionnant les consommations des conteneurs sur cette période, et en les divisant par les capacités maximales des nœuds), puis applique la technique *spread* sur ce nombre réduit de nœuds.
— *heuristic* : L'exécution de l'heuristique de placement pendant la période T_{obs} telle que décrite en Section 3.3 sans exécuter de boucle par la suite.
— *loop* : Notre stratégie d'orchestration complète, soit l'exécution des boucles, après obtention des premières solutions heuristiques, comme décrit en Section 3.4.

4.2.2 Étude de la robustesse

Notre boucle a été développée dans le but de détecter des changements de comportement sur l'infrastructure, et d'ajuster le placement en conséquence. Ces changements peuvent être la conséquence de divers événements :

— Un changement de profil de consommation par un ou plusieurs conteneurs ;
— L'arrivée de nouveaux conteneurs présentant de nouveaux profils de consommation ;
— La disparition de conteneurs (entraînant la perte de profils de consommation).

Dans un contexte métier tel que l'hébergement de micro-services, notre méthode doit être réactive et ajuster la solution de placement rapidement. Par cette étude de *robustesse*, nous confrontons notre méthode aux divers scénarios énumérés ci-dessus, afin d'analyser la performance de son comportement d'adaptation des solutions face aux solutions concurrentes.

De plus, bien que notre méthode doive être réactive aux changements de profil, elle doit aussi montrer une amélioration de la *stabilité* en l'absence de changements (ou en présence de changements minimes). Nous évaluons donc cette stabilité avec des données spécifiques, ne présentant pas ou peu de changement dans les profils observés. Pour ces différents scénarios, nous relevons différents critères :

— $time_{stable}$: le nombre de boucles avant d'obtenir une solution stable permettant d'évaluer la réactivité de la méthode ;

— $moves_{stable}$: le nombre de déplacements observés pendant un régime stable (pas de changement de profil) permettant d'évaluer la stabilité de la méthode ;

Les jeux de données utilisés pour cette étude devant refléter les différents cas cités : pour ce faire, un travail spécifique sur leur génération a été réalisé (voir Section 4.3).

4.2.3 Choix des paramètres

Comme décrit au chapitre précédent, plusieurs paramètres impactent le fonctionnement de notre méthode. Pour chacun d'entre eux, nous avons mené des études préliminaires pour fixer les valeurs ou estimer des intervalles de valeurs de manière empirique pour ces paramètres afin de réaliser ces expérimentations.

4.2.3.1 Paramètres ϵ_C et ϵ_A

Au sein de notre boucle, la pertinence des solutions courantes de clustering et de placement est mesurée par un système de conflits levés par la comparaison des valeurs des variables duales entre deux itérations. Cette comparaison s'effectue grâce à un seuil, appelé respectivement pour le clustering et le placement ϵ_C et ϵ_A. Pour fixer ces seuils, nous avons exécuté notre méthode sur différents jeux de données, en utilisant différentes valeurs, allant de 0.1 à 0.9 par pas de 0.1. Pour chaque couple de valeurs (ϵ_C, ϵ_A), nous avons relevé différents indicateurs mesurant l'impact de ces valeurs sur le graphe de conflit engendré ainsi que sur les modifications de solutions :

— $|V_{G_C}|$: nombre de sommets (donc d'individus différents en conflit) dans le graphe de conflit du clustering G_C ;

— $|E_{G_C}|$: nombre d'arêtes (donc de conflits différents) dans le graphe de conflit du clustering G_C ;

— nb_moves_C : nombre de changements de cluster pour les conteneurs ;

— $|V_{G_A}|$: nombre de sommets (donc d'individus différents en conflit) dans le graphe de conflit du placement G_A ;

— $|E_{G_A}|$: nombre d'arêtes (donc de conflits différents) dans le graphe de conflit du placement G_A ;

— nb_moves_A : nombre de replacements de conteneurs sur les serveurs ;

Ces indicateurs sont récupérés à chaque itération de boucle.

La Figure 4.5 présente les résultats de l'influence des paramètres ϵ_C et ϵ_A sur différents indicateurs :

— *clust_conflict_nodes* : nombre de sommets composant le graphe de conflit pour le clustering ;

— *clust_conflict_edges* : nombre d'arêtes composant le graphe de conflit pour le clustering ;

— *clust_max_deg* : degré maximal des sommets du graphe de conflit pour le clustering ;

— *clust_mean_deg* : degré moyen des sommets du graphe de conflit pour le clustering ;

— *clust_changes* : nombre de changements sur le clustering ;

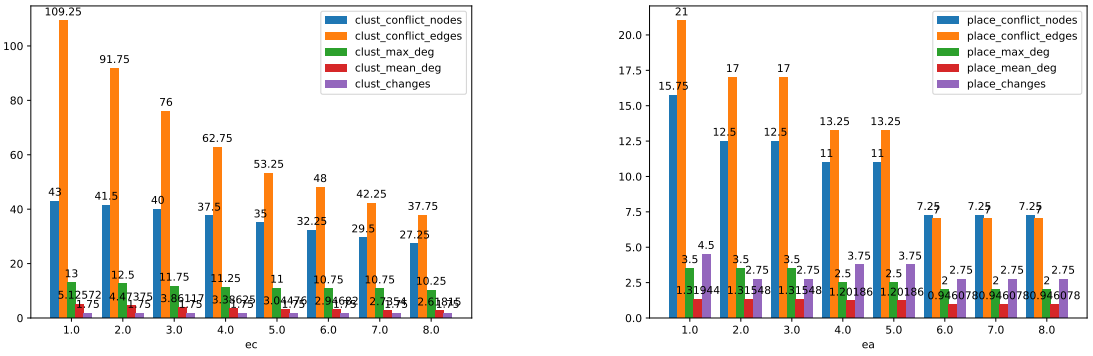
— *place_conflict_nodes* : nombre de sommets composant le graphe de conflit pour le placement ;

— *place_conflict_edges* : nombre d'arêtes composant le graphe de conflit pour le clustering ;

— *place_max_deg* : degré maximal des sommets du graphe de conflit pour le clustering ;

— *place_mean_deg* : degré moyen des sommets du graphe de conflit pour le placement ;

— *place_changes* : nombre de déplacements pour le placement.



(a) ϵ_C (b) ϵ_A

FIGURE 4.5 – Étude de l'influence des paramètres ϵ_C et ϵ_A.

Nous pouvons voir que les valeurs des paramètres ϵ_C et ϵ_A influent beaucoup la taille des graphes de conflits (nombre de sommets et d'arêtes), mais très peu sur le nombre final de modifications, tant pour le clustering que pour le placement. Ceci est dû à l'étape supplémentaire lors des modifications des solutions courantes, qui supprime (de manière gloutonne) les modifications n'améliorant pas la qualité de ces solutions (voir Section 3.4).

4.2.3.2 Paramètre K (nombre de clusters)

Le clustering utilisé dans notre méthode nécessite de fixer le nombre de clusters à utiliser. Ce nombre de clusters, appelé K, est utilisé dans le K-means exécuté dans la période T_{obs} pour obtenir la première solution de clustering, mais également dans les boucles pendant la période T_{run} : K est statique tout au long de l'exécution de notre boucle (plus de détails en Section 3.4). Fixer cette valeur K est un problème classique et de nombreux travaux se sont intéressés à cette problématique. Beaucoup d'entre eux utilisent l'indice *Gap statistic* [Tibshirani et al., 2001] dans une approche statistique pour comparer un indicateur entre le clustering des données considérées et le clustering pour une distribution nulle de référence, autrement dit une distribution sans classe évidente, par exemple en faisant une simulation de Monte Carlo selon une loi uniforme. Cette comparaison est effectuée pour chaque valeur de K pris dans un intervalle donné. L'indicateur utilisé pour comparer les clusterings est la distance intra-cluster, définie comme suit, en considérant D_k la somme des distances entre chaque individu du cluster c_k :

$$W_K = \sum_{k=1}^{K} \frac{1}{2|c_k|} D_k \tag{4.1}$$

À partir de W_K, Gap_K est calculé comme étant la différence entre l'espérance de $\log(W_K^*)$ d'une distribution nulle de référence et $\log(W_K)$ du jeu de données étudié :

$$Gap_K = E^* \log(W_K^*) - \log(W_K) \tag{4.2}$$

Pour un jeu de données fixé, le nombre optimal de clusters K^* est le plus petit K tel que

$$Gap_K \geq Gap_{K+1} - s_{k+1} \tag{4.3}$$

où s_k est l'erreur de simulation pour la génération de la distribution, et calculée à partir de l'écart-type sd_k des B répliques de cette distribution :

$$s_k = \sqrt{1 + \frac{1}{B}} sd_k \tag{4.4}$$

Une étude sur les alternatives de cet indice a été réalisée dans [Mohajer et al., 2011], notamment en considérant directement W_K et non $\log(W_K)$ pour calculer le gap $Gap^*(K)$. Les études ont montré que $Gap^*(K)$ donnait des résultats au moins aussi bons que $Gap(K)$, sauf dans le cas de clusters recouvrants. L'indice $Gap^*(K)$ sera donc utilisé

sur la période T_{obs} de chaque jeu de données, dont la valeur K retournée sera alors utilisée pour exécuter les boucles. Dans la Section 4.3, nous donnons cette valeur de K pour chaque jeu de données décrit.

4.2.3.3 La fenêtre temporelle τ

Le paramètre τ détermine la longueur d'une fenêtre temporelle, c'est à dire le nombre de points de données pris en compte pour exécuter une boucle. Bien que des pistes aient été étudiées pour une meilleure utilisation de ce paramètre (voir Section 5.2), ce paramètre sera fixé de manière empirique, en faisant varier sa valeur sur les différents tests. Sa valeur correspond à un pourcentage de la taille du jeu de données entier : e.g. si $\tau = 10$ et que le jeu de données considéré comporte en tout 50 points de données, alors la fenêtre de temps considérée comportera 5 points de données.

4.2.4 Évaluation du clustering

Notre boucle, en plus de proposer une solution de placement pour conteneurs, définie également une nouvelle méthode de clustering dynamique. Pour évaluer cette dernière, nous avons implémenté deux autres techniques de clustering de l'état de l'art (plus de détails en Section 2.1.2). Nous avons donc 3 résultats de clustering pouvant être comparés :

- *loop_cluster* : clustering utilisé dans notre boucle, comme décrit en Section 3.4 ;
- *kmeans_scratch* : clustering en appliquant la méthode *K-means* sur les données de la fenêtre courante. Ici, le clustering de la boucle précédente n'est donc pas pris en compte, le nouveau clustering est calculé à froid (*from scratch*) ;
- *dynamic_kmeans* : clustering résultant de l'algorithme *StreamKM++*, décrit ci-dessous et très utilisé pour le clustering de données en streaming.

La méthode *StreamKM++* [Ackermann et al., 2012] maintient un aperçu des données grâce à une technique de *merge-and-reduce*. L'étape de fusion (*merge*) est effectuée via une structure de données appelée *bucket set*. L'étape de réduction (*reduce*) utilise une autre structure donnée, appelée *coreset tree*.

Chaque nœud i de l'arbre *coreset tree* est défini par les éléments suivants : un ensemble d'individus E_i, un prototype x_i^p de E_i et la somme des distances au carré SSE_i des objets de E_i dans x_i^p. Cette structure permet de réduire $2m$ objets en m objets et est construite comme suit : d'abord l'arbre ne contient que la racine v, contenant les $2m$ points de données de E_v. Le prototype x_v^p est choisi aléatoirement à partir de E_v. Le calcul de SSE_v est effectué à partir de la définition de x_v^p. Ensuite, deux nœuds enfants de v sont créés : v_1 et v_2. Pour cela, il est nécessaire de choisir un objet de E_v avec une probabilité proportionnelle à $\frac{Dist(x_v^j, x_v^p)^2}{SSE_v}, \forall x_v^j \in E_v$, c'est-à-dire quel'objet le plus distant

de x_v^p a plus de chances d'être choisi. Nous appelons cet objet x_v^q. La prochaine étape est de répartir les objets de E_v vers E_{v_1} ou E_{v_2}, tel que :

$$E_{v_1} = \{x_v^i \in E_v | Dist(x_v^i, x_v^p) < Dist(x_v^i, x_v^q)\} \tag{4.5}$$

$$E_{v_2} = E_v \backslash E_{v_1} \tag{4.6}$$

Cette phase d'expansion de l'arbre s'arrête quand celui-ci contient m feuilles.

Quand un nouveau point de données arrive, il est stocké dans le premier *bucket*. Si ce dernier est plein, toutes ses données sont déplacées vers le second *bucket*. Si celui-ci est également plein, les deux *buckets* sont fusionnés pour donner un ensemble de $2m$ points de données, qui est ensuite réduit à m points de données par la construction d'un *coreset tree* comme détaillé ci-dessus. Ces m points de données sont ensuite stockés dans un troisième *bucket*, sauf si ce dernier est plein, alors une nouvelle étape de *merge-and-reduce* est exécutée.

L'algorithme *K-means* [Arthur and Vassilvitskii, 2006] est finalement utilisé, dans un mode *offline*, sur un ensemble de m points pour obtenir le clsutering final.

Ces différentes méthodes de clustering ont donc été intégrées pour exécuter notre boucle, et plusieurs indicateurs sont relevés pour les comparer et évaluer ces méthodes. Nous séparons ces indicateurs en deux familles : ceux propres au problème de clustering et ceux relevant du placement.

4.2.4.1 Évaluation intrinsèque

Le premier point d'évaluation pour comparer les différentes méthodes de clustering est d'utiliser des indicateurs de performance propres à ce problème de clustering, sans prendre en compte leur influence sur le problème métier.

La question d'évaluation d'un clustering est très étudiée, car il est souvent difficile de mesurer la qualité d'un clustering sans connaissance a priori sur les données évaluées. Plusieurs indicateurs de performance ont été introduits, le plus utilisé étant le coefficient *Silhouette* [Rousseeuw, 1987]. Il est défini pour chaque individu o et prend en compte deux scores :

— a : la distance moyenne entre l'individu o et tous les autres individus appartenant au même cluster ;

— b : la distance moyenne entre l'individu o et tous les autres individus appartenant au cluster le plus proche du cluster auquel appartient o.

Le coefficient Silhouette s_o pour un individu o est défini comme suit :

$$s_o = \frac{b - a}{\max(a, b)} \tag{4.7}$$

Le coefficient Silhouette pour l'ensemble d'un clustering C est défini comme la moyenne des coefficients Silhouette de tous les individus du jeu de données.

4.2.4.2 Évaluation extrinsèque

Les critères ci-dessus nous permettent d'évaluer les méthodes de clustering d'un point de vue propre au domaine de l'apprentissage (non supervisé), mais dans notre cas ce clustering est utilisé dans le but de guider la méthode de mise à jour du placement. Nous disposons donc également d'un critère extrinsèque au clustering pour comparer les différentes méthodes du point de vue de notre problème métier : le placement des conteneurs. Pour ceci, les mêmes critères tels que décrits en Section 4.2.1 sont utilisés : c_1, c_2, c_3 et c_4.

4.2.5 Étude de la performance

En plus des critères métiers cités précédemment, des critères de performance sont également calculés tout au long de l'exécution, afin de mesurer la faisabilité de notre méthode par rapport à la réalité du métier, en considérant les ressources nécessaires (tant au niveau du temps qu'au niveau des ressources physiques). Ces critères sont les suivants :

— *run_time* : temps d'exécution total de la méthode ($T_{obs} + T_{run}$) ;

— *loop_run_time* : temps d'exécution moyen d'une boucle. Ce critère sera discuté en le mettant en relation avec les temps réels présentés par les jeux de données ;

4.3 Présentation des jeux de données étudiés

Tout d'abord, un index des notations utilisées pour présenter toutes les informations concernant les données se trouve en Tableau 4.2.

Notation	Signification
K_G	Nombre de clusters retourné par la méthode *Gap statistic*
Sil.$_{obs}$	Score Silhouette du clustering sur la période T_{obs}
ICS$_{obs}$	Moyenne des distances intra-cluster sur la période T_{obs}
Sil.$_{run}$	Score Silhouette moyen du clustering sur la période T_{run}
ICS$_{run}$	Moyenne des distances intra-cluster sur la période T_{run}

TABLEAU 4.2 – Index des notations utilisées pour les informations des données.

Les données de test sont un élément essentiel de la phase d'évaluation. En effet, elles doivent à la fois refléter le cas d'usage réel du problème attaqué, mais également mettre en avant certains points spécifiques d'évaluation. Nous avons ici deux sources de données de consommation réelles qui sont détaillées dans un premier temps. L'accessibilité aux

données de consommation de conteneurs sur une infrastructure est encore difficile, ce qui limite l'étendue de données issues de consommations réelles pour nos tests. C'est pour cette raison que nous avons également généré des données pour diversifier les profils et les scénarios de placement associés. Le processus de génération de données est détaillé dans un second temps.

Pour chaque jeu de données, nous récoltons diverses informations : générales (taille du problème) et spécifiques au clustering, dans le but d'analyser les profils présents dans les données, ainsi que leur évolution. Le récapitulatif de tous les jeux de données utilisés avec leurs caractéristiques générales (nom, nombre de conteneurs, nombre de serveurs, nombre de points de données) se trouve en Tableau 4.3.

Dataset	Nb conteneurs	Nb nœuds	Nb points de données
AlibabaV1_50n	469	50	144
AlibabaV2_50n	888	50	1344
Alter_way_7d	354	3	973
Profiles_stab_5n	50	5	30
Profiles_add_5n	68	5	30
Profiles_change_5n	71	5	30
Profiles_stab_10n	217	10	50
Profiles_add_15n	477	15	200
Profiles_change_10n	197	10	50

TABLEAU 4.3 – Récapitulatif des caractéristiques générales des jeux de données utilisés.

Pour récolter les informations propres au clustering, nous avons découpé les points de données en deux périodes, qui correspondent aux périodes T_{obs} et T_{run} définies en Section 3.1.1. La période T_{obs} est fixée par défaut aux premiers 20% de l'ensemble des points de données, la période T_{run} aux 80% restants. Nous avons utilisé la période T_{obs} pour appliquer la méthode *Gap statistic* et ainsi fixer K à la valeur K_G retournée pour le jeu de données correspondant.

Ensuite, un clustering (en utilisant cette dernière valeur K_G) est calculé sur la première période T_{obs}, dont nous calculons le score Silhouette, ainsi que la distance moyenne entre individus appartenant au même cluster. Cette distance moyenne est notée ICS et est calculée comme suit :

$$ICS = \frac{1}{I} \sum_{i=1}^{I} \sum_{j \in c_i} w_{i,j} \tag{4.8}$$

Ces mêmes indicateurs sont récupérés (avec la même valeur de K) sur la période T_{run}. La comparaison de ces indicateurs entre les deux périodes nous permet d'évaluer l'évolution des données et des profils pour chaque jeu de données, et ainsi mieux interpréter nos expérimentations. Le tableau récapitulatif des jeux de données et leurs informations relatives au clustering se trouve Tableau 4.4.

Dataset	K_G	Sil.$_{obs}$	ICS$_{obs}$	Sil.$_{run}$	ICS$_{run}$
AlibabaV1_50n	2	0.94	787.05	0.92	1033.12
AlibabaV2_50n	2	0.96	9146.20	0.97	7691.87
Alter_way_7d	3	0.96	2.90e^{11}	0.96	2.46e^{11}
Profiles_stab_5n	3	0.91	1.62	0.91	1.52
Profiles_add_5n	4	0.64	16.64	0.62	17.28
Profiles_change_5n	3	0.73	15.80	0.56	19.71
Profiles_stab_10n	6	0.61	22.20	0.65	20.47
Profiles_add_15n	6	0.59	1010.83	0.45	1301.90
Profiles_change_10n	5	0.57	29.06	0.55	28.93

TABLEAU 4.4 – Récapitulatif des informations sur les profils des jeux de données utilisées.

4.3.1 Jeux de données réels

4.3.1.1 Alter_way_7d

En interne au sein d'Alter way, des données provenant de trois nœuds pour 354 conteneurs sont remontées en continu. Un condensé d'une semaine de ces données a permis l'obtention d'un jeu de données, dont un aperçu des consommations de CPU des conteneurs est visible en Figure 4.6.

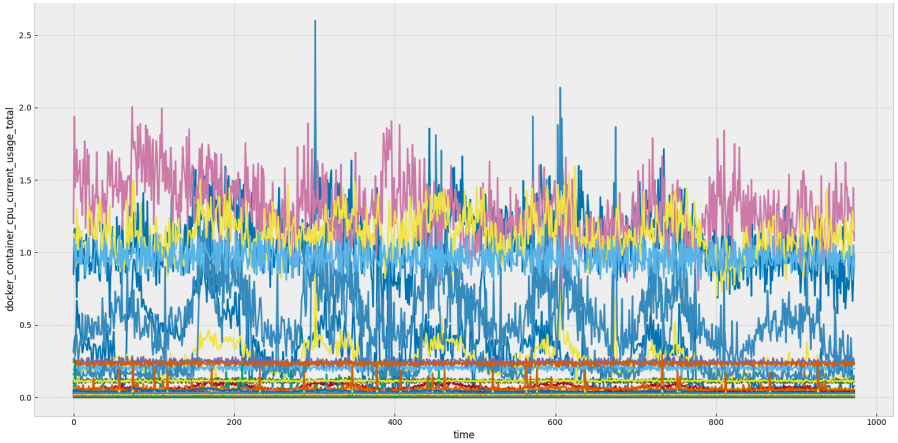


FIGURE 4.6 – Consommation de CPU du dataset d'Alter way

4.3.1.2 Alibaba

Alibaba a mis en ligne des données de consommation d'une partie de leur infrastructure [Alibaba, 2017a], à travers un premier jeu de données représentant environ 1100 nœuds pour plus de 10 000 conteneurs sur une durée de 24 heures. Un an plus tard, un deuxième jeu de données a été publié, représentant près de 2600 nœuds pour 50 000 conteneurs et une durée de 8 jours. Pour des raisons techniques, nous avons dans un premier temps construit un sous-ensemble de données de chacun de ces jeux de données, gardant les données de consommation de 50 nœuds (choisis aléatoirement) sur les mêmes durées (respectivement 24 heures et 8 jours). Ces jeux de données sont nommés

AlibabaV1_50n et AlibabaV2_50n, tirés respectivement de la première publication de données [Alibaba, 2017b] et de la deuxième publication de données [Alibaba, 2018].

4.3.1.3 Spécificités des jeux de données réels

Dans le Tableau 4.4, montrant les informations de clustering relevées sur les jeux de données, nous relevons plusieurs remarques :

1. L'utilisation de 2 ou 3 clusters suffit à obtenir un clustering des données de qualité (selon le score Silhouette calculé, variant entre 0.94 et 0.97). Ceci reflète des profils de consommation peu distincts.
2. Le clustering utilisé en début de jeux de données (période T_{obs}) génère un score Silhouette du même ordre que celui sur les données de la période T_{run}, ce qui reflète des profils de consommation stables dans le temps (pas ou peu de changements de profil au fil du temps).

Ces spécificités s'expliquent en grande partie par l'accessibilité difficile aux données de consommation (l'infrastructure entière et l'ensemble des conteneurs présents ne sont pas accessibles), ce qui réduit la diversité des données finalement observées.

4.3.2 Jeux de données synthétiques

Comme mentionné dans la découverte des jeux de données réels, ces derniers présentent peu de cas de changements dans les données (e.g. changements de profils de consommation). Pour évaluer notre méthode et sa résistance à des données plus perturbées, ainsi que mieux refléter la diversité des données de consommation possible sur les infrastructures, nous avons généré de nouveaux jeux de données, dont nous détaillons la création et leur motivation en terme d'évaluation. Un générateur de données a été développé pour cela, permettant de renseigner des profils types spécifiques et de générer de nouvelles données à partir de ces profils, en y apportant des modifications diverses. Les profils types utilisés pour générer les différentes données sont illustrés en Figure 4.7. Bien que le générateur ait été développé avec la possibilité de renseigner n'importe quel nombre de profil type, nous n'en utilisons seulement 2 ici car ceux-ci, en y appliquant toutes les transformations possibles décrites ci-dessous, nous permettent de générer une diversité suffisante de profils pour réaliser nos expérimentations.

Ces profils types sont définis par différents points de données. Chaque point de donnée est utilisé comme valeur initiale pour générer un point de donnée du nouveau profil à générer. Pour générer ce nouveau point de donnée o_t, à partir des points de données $\overline{o_t}$ du profil type, plusieurs facteurs de transformation sont possibles :

— Une translation verticale $translate$: $o_t = \overline{o_t} + translate$;

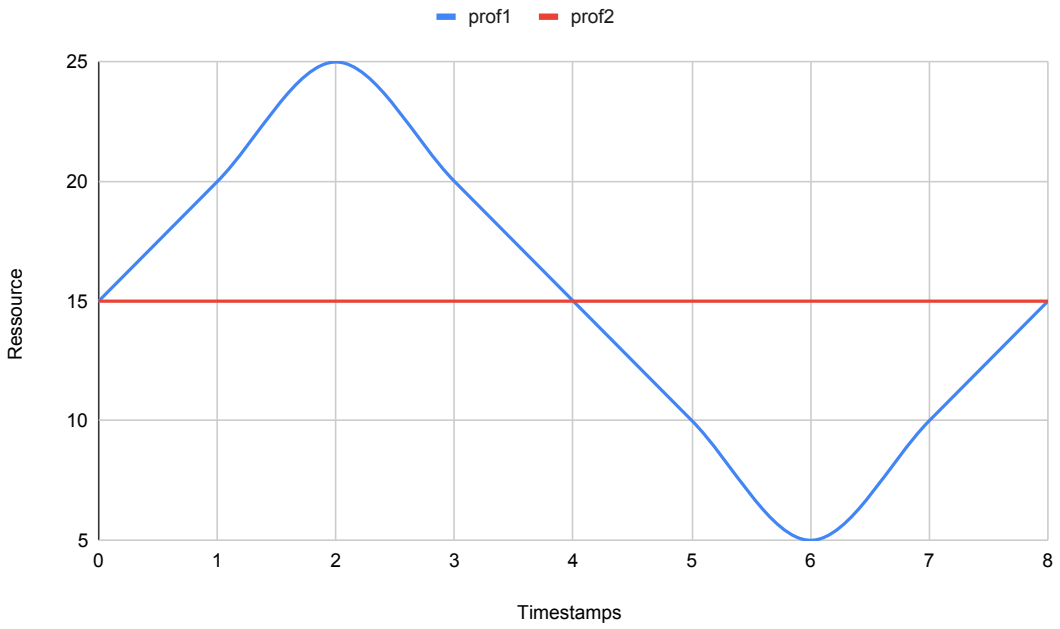


FIGURE 4.7 – Profils types utilisés dans le générateur de données.

Algorithme 13 : Génération d'un profil de consommation

Input : Profil type $\overline{o}$, $translate$, $delay$, $coef$, $ampli$, t_{min}, t_{max}
Output : Profil o

1 **for** $t = t_{min}$; $t <= t_{max}$; $t++$ **do**
2 $\quad val_{init} = (\overline{o_{t+delay}} + translate) \times coef$
$\quad o_t = random(val_{init} \times (1 - ampli), val_{init} \times (1 + ampli))$

— Une translation horizontale $delay$: $o_t = \overline{o_{t+delay}}$;

— Un coefficient multiplicateur $coef$: $o_t = \overline{o_t} \times coef$;

— Une taille de période $size\_period$, indiquée par le nombre de points de données compris dans la période du profil type.

La génération de tous les points de données, au regard des transformations possibles citées ci-dessus, est présenté en Algorithme 13. À l'échelle du profil entier à générer, celui-ci peut subir diverses opérations à partir d'un certain point de donnée afin d'apporter du changement :

— le profil du conteneur suit un même profil type pour toute la durée du dataset ;

— le profil du conteneur suit un profil type, puis suit un autre profil (déjà existant dans les données ou non) type à partir d'un certain point de donnée ;

— le profil du conteneur suit un profil type puis devient nul (les conteneurs en question deviennent inactifs en cours d'exécution) ;

— le profil du conteneur suit une consommation nulle puis suit un profil type (les conteneurs en question deviennent actifs en cours d'exécution).

Dans le générateur, pour appliquer l'un de ces changements à un point de donnée t_{change} sur un jeu entier comprenant des données entre $t_{D_{min}}$ et $t_{D_{max}}$, il suffit d'appliquer l'Algorithme 13 avec un certain profil type avec $t_{min} = t_{D_{min}}$ et $t_{max} = t_{change}$, puis de l'appliquer de nouveau avec un autre profil type et $t_{min} = t_{change}$ et $t_{max} = t_{D_{max}}$.

En guise d'exemple, nous détaillons par la suite la création de plusieurs jeux de données générés grâce à ce générateur. Chacun de ces jeux de données comporte l'indication "Xn" dans son nom : "X" indique le nombre de nœuds générés pour ce jeu de données.

4.3.2.1 Création de "Profiles_stab_5n"

Pour générer le jeu de données "Profiles_stab_5n", nous créons trois profils types, à partir des profils d'origine illustrés en Figure 4.7 :

- un profil p_1 suivant le profil d'origine *prof1*, sans transformation ;
- un profil p_2 suivant le profil d'origine *prof2*, en appliquant une translation verticale ($translate = -14$) ;
- un profil p_3 suivant le profil d'origine *prof1*, en appliquant une translation horizontale ($delay = +4$).

Le paramètre *size_period* est volontairement fixé à une même valeur ($size\_period = 8$), ainsi les profils p_1 et p_3 sont créés de telle sorte qu'ils présentent des profils symétriquement opposés l'un à l'autre. Les individus sont créés en suivant un des profils cités ci-dessus, en générant chaque point de donnée $o_t = random(val_{p,t} \times (1 - ampli), val_{p,t} \times (1 + ampli))$, avec $val_{p,t}$ la valeur du point de donnée du profil p au temps t. De cette manière, 8 conteneurs ont été créés en suivant le profil p_1, 35 en suivant le profil p_2, puis 7 en suivant le profil p_3. Le nombre d'individus créés suivant les profils p_1 et p_2 est volontairement du même ordre de grandeur, afin d'avoir des individus en opposition de phase de manière équilibrée. Le nombre d'individus suivant le profil p_3 est lui aléatoire. Aucun changement de profil n'a été appliqué dans ce jeu de données, servant de référence pour des données très stables en terme de variance de charge. En effet, un tel jeu de données nous permet d'analyser le comportement de notre méthode dans un contexte d'absence de modification des profils de consommation, notamment en évaluant la stabilité des solutions. La création de profils en opposition de phase devrait impacter les critères c_2 et c_3 selon la méthode de placement utilisée. Un aperçu de ce jeu de données est fourni en Figure 4.8.

En Tableau 4.4 présentant les informations liées aux profils de consommation et à leur évolution pour l'ensemble des jeux de données, nous voyons que concernant le jeu "Profiles_stab_5n", sa stabilité est illustrée par le score Silhouette identique entre les périodes T_{obs} et T_{run}, ainsi que les distances moyennes intra-clusters, qui restent très proches d'une période à une autre. Le jeu de données "Profiles_stab_10n" a été créé

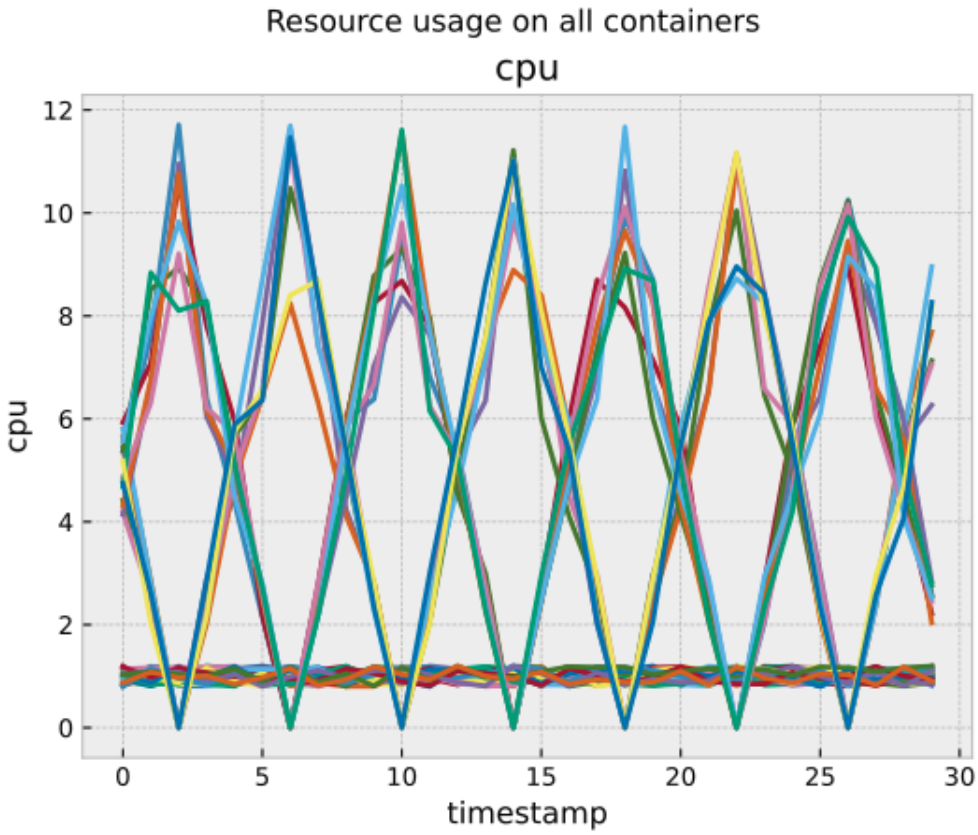


FIGURE 4.8 – Evolution de consommation pour le jeu de données "Profiles_stab_5n".

en suivant le même principe de stabilité des données, mais en augmentant la taille de celles-ci : le nombre de conteneurs passe de 50 à 217, le nombre de nœuds de 5 à 10 et le nombre de profils utilisés passe de 3 à 8. Ce nouveau jeu de données plus grand permet la même évaluation que le jeu "Profiles_stab_5n" mais en considérant le passage à l'échelle (plus d'individus, plus de nœuds et plus de profils).

4.3.2.2 Création de "Profiles_change_5n"

Afin de montrer comment la diversité au sein d'un jeu de données est générée, nous prenons l'exemple du jeu d'essai "Profiles_change_5n".

Ce jeu d'essai a été créé dans le but d'analyser le comportement de notre méthode dans le cas de modifications dans les profils de consommation sans apparition ou disparition de profils (les individus concernés passent d'un profil à un autre profil déjà existant).

Pour ce jeu d'essai, 5 profils types, visibles en Figure 4.9, sont créés selon les caractéristiques décrites en Tableau 4.5.

Pour amener du changement dans l'évolution des consommations, le générateur intègre des changements de profils pour certains individus. Le nombre de changements à intégrer est fixé en paramètre, et pour chacun de ces changements le générateur fixe aléatoirement différents paramètres :

Profil id	1	2	3	4	5
Profil type	*prof1*	*prof1*	*prof2*	*prof1*	*prof1*
delay	11	18	3	4	23
translate	21,5	17,3	40,6	36,5	31,7
coef	21,3	48,6	45,9	43,1	2,9
ampli	0,23	0,36	0,23	0,16	0,31
Nb individus	20	7	19	11	19

TABLEAU 4.5 – Liste des caractéristiques utilisées pour la création des 5 profils du jeu de données "Profiles_change_5n".

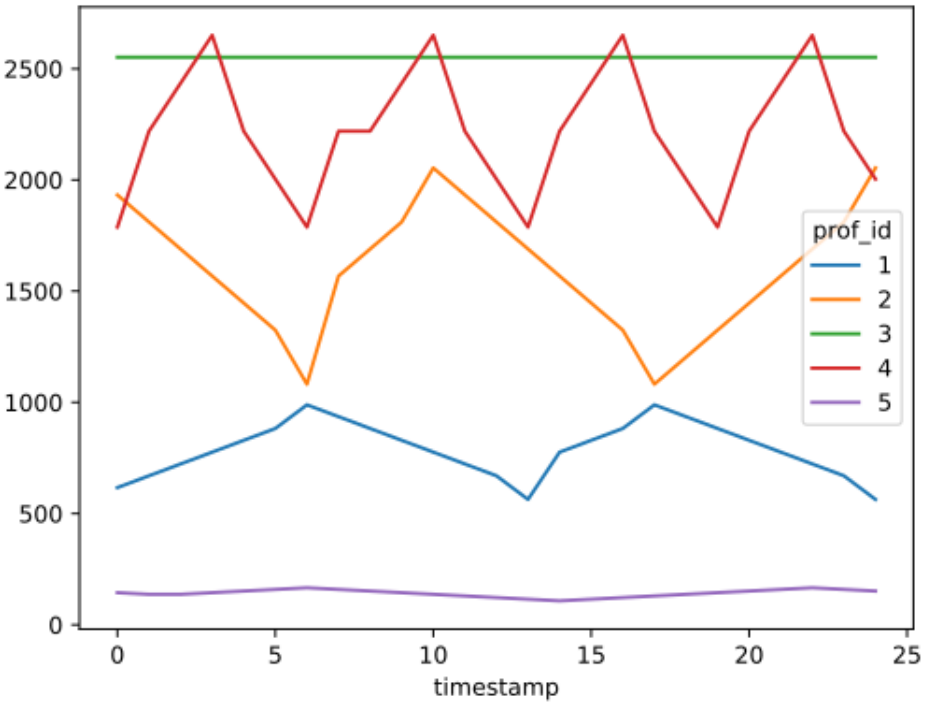


FIGURE 4.9 – Profils types créés pour le jeu "Profiles_change_5n".

— le temps t_{change} auquel le changement s'applique ;

— le nombre d'individus concernés par ce changement ;

— le profil old_prof suivi par les individus jusqu'à t_{change} ;

— le profil new_prof suivi par les individus à partir de t_{change}.

Dans cet exemple, un seul changement est intégré : au temps $t_{change} = 32$, 10 individus passent du profil d'identifiant 4 au profil d'identifiant 3. Un aperçu de ce jeu de données est donné en Figure 4.10.

Dans le Tableau 4.4, la différence du score Silhouette calculé pour le jeu "Profiles_change_5n" entre les périodes T_{obs} et T_{run} reflètent les modifications de profil de consommation pour ce jeu d'essai : le clustering calculé pendant la période T_{obs} n'est plus pertinent

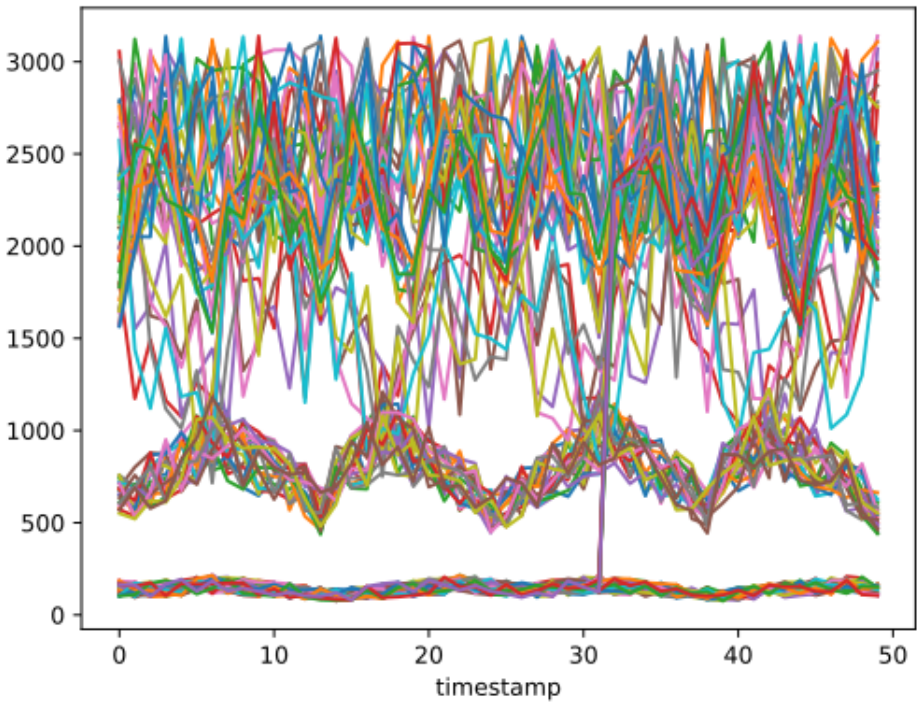


FIGURE 4.10 – Evolution de consommation pour le jeu de données "Profiles_change_5n".

pour la période T_{run}. Le jeu de données "Profiles_change_10n" a été créé en suivant le même principe de modification de profils par des individus, mais en augmentant la taille des données : 197 individus au lieu de 71, 10 nœuds au lieu de 5 et 9 profils utilisés. Par ailleurs, dans ce nouveau jeu de données, plusieurs changements (trois) sont intégrés à des temps différents.

4.3.2.3 Création de "Profiles_add_5n"

Alors que le jeu de données décrit ci-dessus permet de considérer un scénario dans lequel les individus passent d'un profil existant à un autre profil également existant, d'autres jeux de données sont créés afin d'étudier le comportement de notre méthode dans le cas où certains individus changent de profils de consommation, mais adopte un nouveau profil éloigné des profils déjà identifiés. C'est dans ce but que le jeu "Profiles_add_5n" a été généré.

Pour effectuer ces changements dans le jeu de données, le générateur utilise les mêmes indicateurs décrits pour le jeu "Profiles_change_5n", mais en générant un nouveau profil pour new_prof au lieu de le sélectionner parmi ceux existant :

— le temps t_{change} auquel le changement s'applique ;

— le nombre d'individus concernés par ce changement ;

— le profil old_prof suivi par les individus jusqu'à t_{change} ;

— le profil new_prof, généré avec de nouveaux paramètres, suivi par les individus à partir de t_{change}.

Le Tableau 4.6 montre les paramètres générés pour créer les différents profils (les 5 profils présents dès le début ainsi que le profil apparaissant en cours d'exécution). Ce nouveau profil (d'identifiant 6 sur le tableau) présente d'ailleurs un nombre d'individus nul, car les paramètres présentent l'état au début du jeu de données. La modification de profil a été générée au temps $t_{change} = 35$, où 8 individus passent du profil d'identifiant 1 au profil 6.

Profil id	1	2	3	4	5	6
Profil type	*prof1*	*prof1*	*prof2*	*prof1*	*prof1*	*prof1*
delay	19	8	7	2	19	14
translate	8,1	9,1	46,8	-5,0	15,3	47,6
coef	25,8	20,9	7,6	23,7	27,3	35,1
ampli	0,11	0,25	0,15	0,32	0,35	0,15
Nb individus	10	16	7	10	15	0

TABLEAU 4.6 – Liste des caractéristiques utilisées pour la création des 6 profils du jeu de données "Profiles_add_5n".

Les profils types ainsi créés (y compris le nouveau profil ajouté en cours d'exécution) sont visibles en Figure 4.11.

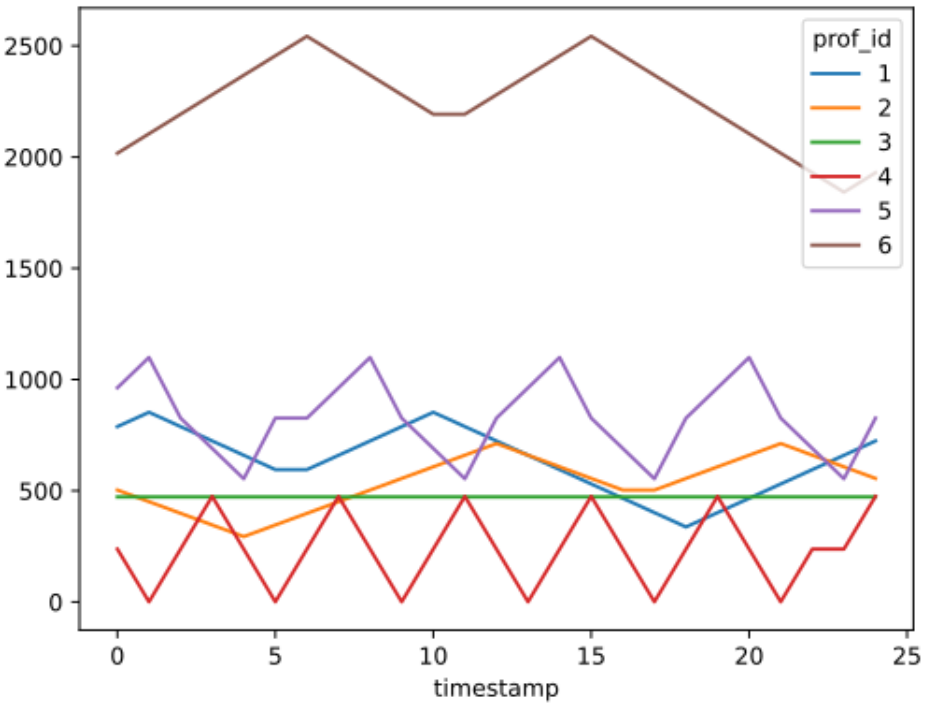


FIGURE 4.11 – Profils types créés pour le jeu "Profiles_add_5n".

Un aperçu de ce jeu de données est donné en Figure 4.12.

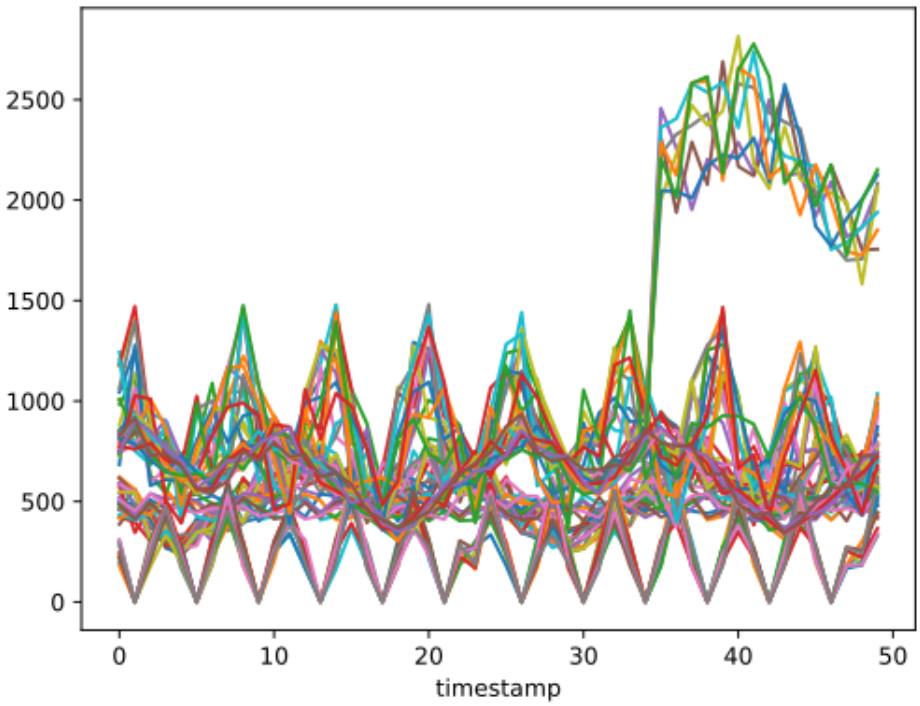


FIGURE 4.12 – Evolution de consommation pour le jeu de données "Profiles_add_5n".

De la même manière que pour le jeu "Profiles_change_5n", la différence des scores Silhouette calculés sur les périodes T_{obs} et T_{run} pour le jeu "Profiles_add_5n" (visibles dans le Tableau 4.4) montre un premier clustering non conforme pour la suite du jeu de données. Le jeu de données "Profiles_add_15n" a été créé en suivant les mêmes changements, mais en augmentant la taille des données : 417 individus, 15 nœuds et 11 profils utilisés. Par ailleurs, ce jeu de données est le plus important (en termes de taille) parmi les jeux de données synthétiques.

4.4 Résultats et analyse des expérimentations

Pour présenter les résultats, différents supports sont utilisés. Tout d'abord, les tableaux présentent, sauf mention contraire, les valeurs moyennes pour des exécutions utilisant différentes valeurs de τ comprises entre 5 et 25 par pas de 5.

Par ailleurs, des profils de performance sont également utilisés tout au long de cette section. Les profils de performance [Dolan and More, 2002] montrent la probabilité qu'une méthode donne une solution à une distance x de la meilleure solution pour toutes les méthodes considérées. Étant donné un ensemble de problèmes P et un ensemble d'algorithmes S, on définit par $c_{s,p}$ l'efficacité de la méthode $s \in S$ à résoudre le problème $p \in P$. L'efficacité peut être liée au temps de résolution ou à un critère donné.

Point d'évaluation	Instances	Critères	Motivation
Paramètres ϵ_C et ϵ_A	Tous (non)	$\|V_{G_C}\|$, $\|E_{G_C}\|$, $\|V_{G_A}\|$, $\|E_{G_A}\|$, nb_movesA	Fixer les paramètres ϵ_C et ϵ_A
Clustering intrinsèque	Tous	*Silhouette*	Évaluer la qualité intrinsèque clustering
Clustering extrinsèque	Tous	c_1, c_2, c_3, c_4	Évaluer la qualité clustering via placement
Robustesse	*"Profiles*"*		Évaluer réactivité et stabilité de la méthode
Efficacité globale	Tous	c_1, c_2, c_3, c_4	Évaluer la qualité du placement tout au long de l'exécution (par critères métier)
Performance technique	Tous	$run\_time, loop\_run\_time$	Évaluer l'utilisabilité réelle de la méthode

TABLEAU 4.7 – Récapitulatif des tests réalisés.

Dans notre cas, $c_{s,p}$ se définit en fonction des valeurs des critères métiers calculées pour chaque méthode. Nous définissons $c_{min,p}$ la valeur de la méthode la plus efficace pour le problème p, tel que :

$$c_{min,p} = \min_{s \in S} c_{s,p} \tag{4.9}$$

L'efficacité relative d'une méthode s pour un problème p donné, appelée *performance ratio* et notée $r_{s,p}$, est ensuite définie comme suit :

$$r_{s,p} = \frac{c_{s,p}}{c_{min,p}} \tag{4.10}$$

Finalement la fonction de performance ρ_s d'une méthode s est définie comme suit :

$$\rho_s(t) = \frac{\#\{p \in P | r_{s,p} \leq t\}}{\#P} \tag{4.11}$$

Concrètement, $\rho_s(t)$ représente la probabilité pour une méthode s d'être t fois moins efficace que les meilleures méthodes possibles. Ce qui signifie que si $\rho_{s,1}$ est proche de 1, alors la méthode s est parmi les plus efficaces.

Notons que l'ensemble des expérimentations a été réalisé sur une machine virtuelle équipée de 24 CPU, de type Intel(R) Xeon(R) Gold 6148, avec une fréquence de 2.40GHz, et possédant 100GB de RAM.

4.4.1 Efficacité globale de la méthode

Le Tableau 4.8 recense les notations utilisées pendant cette partie dédiée à l'efficacité globale de la méthode, vis-à-vis des critères métiers identifiés.

Notation	Signification
c_1	Nombre de nœuds
c_2	Amplitude maximale de consommation sur les nœuds
c_3	Variance maximale de consommation sur les nœuds
c_4	Nombre d'alertes engendrées

TABLEAU 4.8 – Index des notations utilisées pour l'évaluation de l'efficacité globale.

4.4.1.1 Jeux de données réels

Le Tableau 4.9 présente la comparaison des résultats pour les critères c_1, c_2, c_3 et c_4, relevés pour chaque jeu de données réel (les jeux de données d'Alibaba et d'Alter way) en appliquant chacune des méthodes décrites en Section 4.2.1. Les critères c_1 et c_4 sont indiqués par leur valeur brute (respectivement le nombre de nœuds et le nombre d'alertes). En revanche, pour simplifier la lecture des tableaux, les critères c_2 et c_3 sont représentés de deux différentes manières :

— la colonne *[0 ;100]* : les valeurs brutes sont normalisées sur l'intervalle $[0; 100]$. Ceci signifie que pour chaque jeu de données, la méthode présentant la plus petite valeur (val_{min}) pour l'un de ces deux critères sera représentée par $0,00$ dans le tableau, et celle présentant la plus grande valeur (val_{max}) pour le même critère sera représentée par $100,00$. La valeur normalisée $val_{norm}(val)$ à partir de la valeur brute val est calculée de la manière suivante :

$$val_{norm}(val) = \frac{val - val_{min}}{val_{max} - val_{max}} \times 100 \tag{4.12}$$

— la colonne *%err* : les valeurs sont utilisées pour calculer leur pourcentage d'erreur par rapport à la meilleure valeur val_{min} pour ce critère. Ce pourcentage $val_{err}(val)$ à partir d'une valeur brute val est calculée de la manière suivante :

$$val_{err}(val) = \frac{val - val_{min}}{val_{min}} \times 100 \tag{4.13}$$

De plus, pour chaque jeu de données et chaque critère considéré, la méthode présentant le meilleur résultat est donnée en **gras**.

Le premier critère métier c_1, représentant le nombre de nœuds, est amélioré en utilisant les méthodes *IC*, l'heuristique seule *heur* et notre *boucle* (méthode *loop*). La différence entre notre méthode *loop* et les deux autres méthodes économisant des nœuds s'explique

Dataset	Méthode	c_1	c_2		c_3		c_4
		nb nœuds	*ampli_max*		*var_max*		*nb alertes*
			[0 ;100]	*%err*	*[0 ;100]*	*%err*	
AlibabaV1_50n	*initial*	50	**0,00**	**0,00**	**0,00**	**0,00**	**0**
	spread	50	46,18	16,44	46,68	33,14	**0**
	iter-consol	46,8	100	35,13	100	47,64	3,21
	heuristic	46,3	1,06	1,15	14,26	3,19	2,73
	loop	**46**	1,06	1,15	3,72	0,72	0,20
AlibabaV2_50n	*initial*	50	**0,00**	**0,00**	**0,00**	**0,00**	**0**
	spread	50	0,90	0,88	0,64	6,40	**0**
	iter-consol	44,9	100	97,47	100	227,20	6,7
	heuristic	44,4	68,35	52,02	3,69	48,51	4,8
	loop	**44**	47,22	21,08	1,53	14,68	1,4
Alter_way_7d	*initial*	3	6,78	2,23	8,32	1,84	**0**
	spread	3	**0,00**	**0,00**	**0,00**	**0,00**	**0**
	iter-consol	**2**	100	32,65	100	11,70	2,85
	heuristic	**2**	61,27	9,03	15,50	7,36	1,69
	loop	**2**	15,19	4,01	9,32	2,45	0,15

TABLEAU 4.9 – Résultats des critères c_1 (nombre de nœuds), c_2 (amplitude maximale de consommation), c_3 (variance maximale de consommation) et c_4 (nombre d'alertes) sur les jeux de données réels (Alibaba et Alter way).

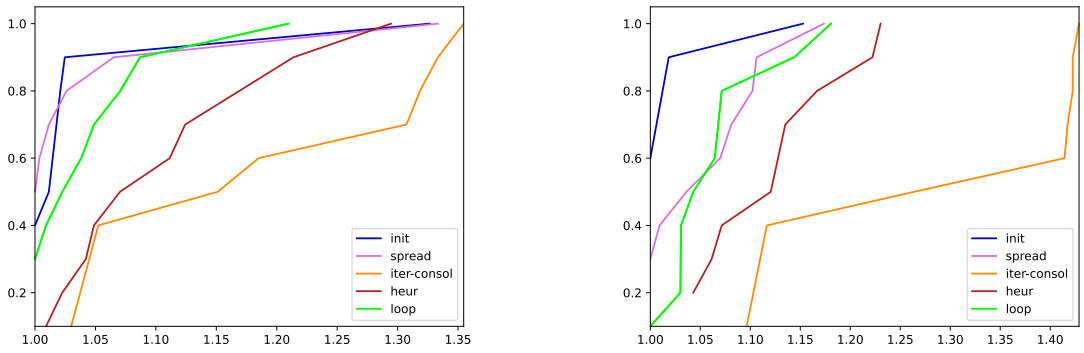
par le fait que ces dernières, sans ajustement du placement au cours de temps, ont dû utiliser un serveur supplémentaire pour répondre aux alertes levées.

Ces alertes, représentées par le critère c_4, sont également plus nombreuses pour les méthodes *IC* et *heur* par rapport à notre méthode *loop*. Nous voyons que, en utilisant plus de nœuds, les méthodes *initial* et *spread* ont moins de chance de déclencher une alerte liée aux capacités maximales des nœuds. Cependant, notre boucle fournit les informations supplémentaires pour gérer ces risques d'*alerte*, grâce à un ajustement *en ligne* du placement qui prend en compte l'évolution des profils. C'est pourquoi, bien qu'il soit difficile d'améliorer ce critère c_4 par rapport aux deux premières méthodes représentées, on peut tout de même garder un très faible nombre d'alertes avec notre boucle par rapport aux autres méthodes économisant des nœuds.

Les critères de stabilité, représentés par c_2 et c_3, sont également dominés par les méthodes *initial* et *spread* en utilisant plus de nœuds. Là aussi, nous remarquons que notre *loop* présente des critères de stabilité proches des deux méthodes citées ci-dessus, tout en dominant les deux autres méthodes économisant des nœuds.

Les profils de performance nous permettent une analyse plus détaillée de ces critères de stabilité. En effet, les valeurs des critères c_2 et c_3 étant fortement dépendantes des valeurs de consommation, dont les niveaux varient fortement d'un jeu de données à un autre, nous proposons en Figure 4.13 un résumé des résultats de ces deux critères via

les profils de performance pour les jeux de données réels (Alibaba et Alter way).

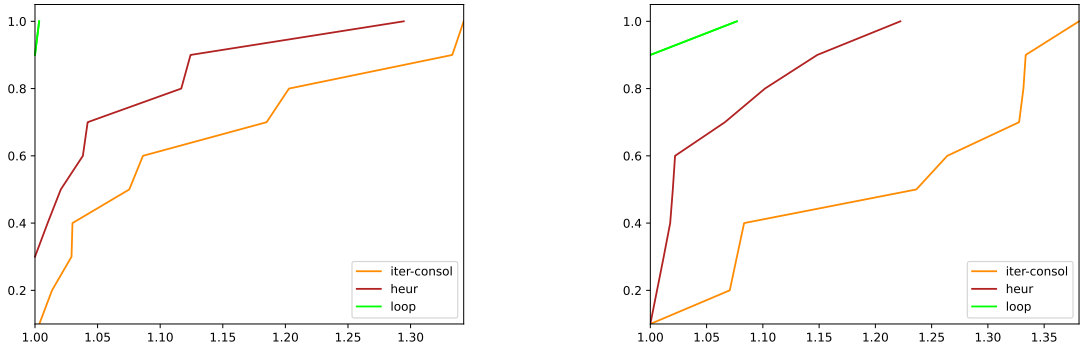


(a) Critère c_2 (b) Critère c_3

FIGURE 4.13 – Comparaison des critères c_2 et c_3 avec toutes les méthodes, sur les jeux de données réels.

Ces profils de performance confirment la domination des méthodes *initial* et *spread* sur les autres méthodes concernant les critères de stabilité de consommation. Or il est difficile de comparer ces critères lorsque l'on sait que ces deux méthodes utilisent plus de nœuds. C'est pourquoi nous proposons deux analyses plus détaillées de ces critères :

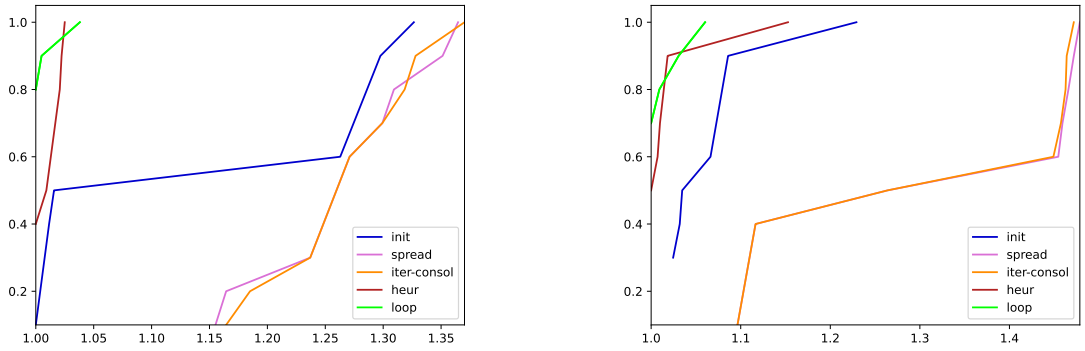
— Dans un premier temps, nous comparons les critères de stabilité entre les méthodes optimisant un autre critère crucial : le nombre de nœuds c_1. La Figure 4.14 montre donc les mêmes profils de performance en ne comparant que les méthodes économisant des nœuds (donc les méthodes *IC*, *heur* et *loop*). Nous voyons la domination de notre méthode *loop*, ce qui est en phase avec le critère de nombre d'alertes c_4 : en améliorant la stabilité de la consommation sur les nœuds, nous réduisons le risque de surcharge.



(a) Critère c_2 (b) Critère c_3

FIGURE 4.14 – Comparaison des critères c_2 et c_3 avec les méthodes économisant le nombre de nœuds, sur les jeux de données réels.

— Une seconde comparaison est effectuée entre toutes les méthodes, mais en fixant le critère c_1 pour toutes ces méthodes : ainsi les critères de stabilité peuvent être comparés dans des contextes métier similaires. La Figure 4.15 montre les profils de performance pour les critères c_2 et c_3 en fixant le nombre de nœuds au maximum de nœuds utilisés parmi toutes les méthodes. Dans ce cas, le critère de stabilité est privilégié et notre méthode domine toutes les autres, y compris le placement initial résultant de l'action d'administrateurs systèmes.



(a) Critère c_2 (b) Critère c_3

FIGURE 4.15 – Comparaison des critères c_2 et c_3 avec toutes les méthodes en fixant le nombre de nœuds, sur les jeux de données réels.

4.4.1.2 Jeux de données synthétiques

Le même tableau de résultats globaux, mais présentant les résultats pour les jeux de données générés, se trouve Tableau 4.10. La méthode *initial* n'est ici pas présente, car les données ayant été générées il n'y pas de placement résultant d'une expertise (e.g. administrateur système) contrairement aux jeux de données réels.

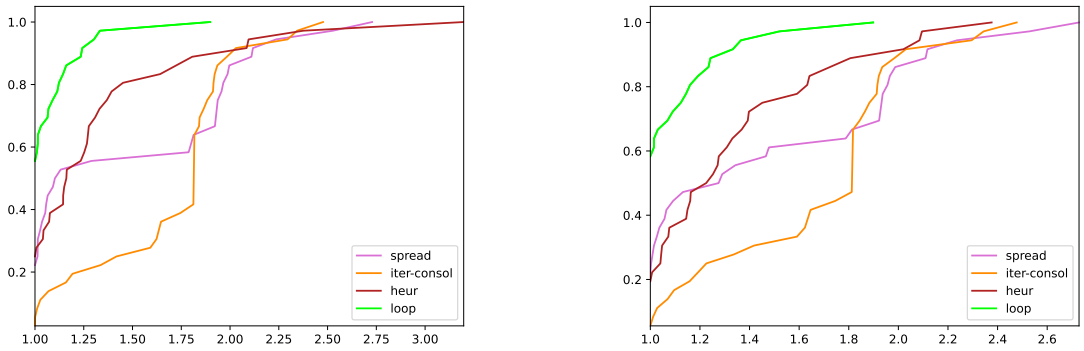
Par rapport aux jeux de données réels, ici la marge de manoeuvre liée au nombre de nœuds est plus faible : les méthodes économisant le nombre de nœuds présentent des valeurs de critère c_1 très proches voire égales à celles de la méthode *spread*.

Le critère du nombre de nœuds étant plus homogène entre les méthodes, nous observons plus facilement de meilleurs résultats pour les critères de stabilité avec notre *loop*. Les profils de performance générés à partir des résultats des critères c_2 et c_3, présents en Figure 4.16, confirment cette observation où la méthode *loop* domine globalement toutes les autres méthodes, y compris la méthode *spread* utilisant plus de nœuds.

Il est intéressant de noter que pour les jeux de données Profiles_stab_5n et Profiles_stab_10n, la méthode *heur* donne des résultats quasi identiques que la boucle *loop* : ceci s'explique par le fait que ces deux jeux de données ont été générés sans modifications de consommation (les individus présentent le même profil tout le long du jeu de données).

Dataset	Méthode	c_1 *nb nœuds*	c_2 *ampli_max*		c_3 *var_max*		c_4 *nb alertes*
			[0;100]	*%err*	*[0;100]*	*%err*	
Profiles_stab_5n	*spread*	**5**	100,00	23,43	100,00	93,69	1
	iter-consol	**5**	97,02	20,09	91,35	59,12	**0**
	heuristic	**5**	**0,00**	**0,00**	**0,00**	**0,00**	**0**
	loop	**5**	**0,00**	**0,00**	**0,00**	**0,00**	**0**
Profiles_add_5n	*spread*	5	**0,00**	**0,00**	3,85	6,74	**0**
	iter-consol	5	100,00	62,46	100,00	78,46	6,3
	heuristic	5	41,38	14,66	54,40	18,89	4,1
	loop	**4,9**	0,97	4,06	**0,00**	**0,00**	1,7
Profiles_change_5n	*spread*	**5**	100,00	129,51	100,00	110,03	**0**
	iter-consol	**5**	100,00	129,51	100,00	110,03	11,7
	heuristic	**5**	14,07	36,83	11,28	17,37	4,5
	loop	**5**	**0,00**	**0,00**	**0,00**	**0,00**	1,2
Profiles_stab_10n	*spread*	10	2,21	9,17	3,13	3,73	**0**
	iter-consol	**8,7**	100,00	98,37	100,00	91,26	3,9
	heuristic	**8,7**	**0,00**	**0,00**	3,74	5,25	3,4
	loop	**8,7**	1,19	3,72	**0,00**	**0,00**	2,9
Profiles_add_15n	*spread*	**15**	100,00	66,69	100,00	61,85	**0**
	iter-consol	**15**	100,00	66,69	100,00	61,85	22,3
	heuristic	**15**	63,51	43,07	39,10	38,91	1,8
	loop	**15**	**0,00**	**0,00**	**0,00**	**0,00**	1,8
Profiles_change_10n	*spread*	10	10,21	3,63	2,54	8,45	**0**
	iter-consol	9,6	100,00	37,91	100,00	56,62	2,3
	heuristic	9,5	34,65	11,55	23,40	14,70	0,2
	loop	**9,3**	**0,00**	**0,00**	**0,00**	**0,00**	0,2

TABLEAU 4.10 – Résultats des critères c_1 (nombre de nœuds), c_2 (amplitude maximale de consommation), c_3 (variance maximale de consommation) et c_4 (nombre d'alertes) sur les jeux de données synthétiques.



(a) Critère c_2

(b) Critère c_3

FIGURE 4.16 – Comparaison des critères c_2 et c_3 avec toutes les méthodes, sur les jeux de données synthétiques.

La méthode *heur* appliquant les mêmes méthodes de placement pendant la période T_{obs} et la boucle étant très stable, les résultats de placement sont les mêmes. En revanche, nous observons des alertes pour le jeu de données Profiles_stab_10n, celles-ci sont dûes à l'économie de nœuds réalisée, avec un niveau de consommation total sur ceux-ci étant proches de la limite maximale.

En ce qui concerne les jeux de données générés avec des changements poendant la période T_{run}, soit les jeux de données Profiles_add_5n, Profiles_change_5n, Profiles_-add_15n et Profiles_change_10n (voir Section 4.3 pour plus de détails sur ceux-ci), les critères de stabilité (c_2 et c_3) sont dominés par notre méthode *loop* : si aucune alerte de consommation maximale n'est levée, les autres méthodes n'adapteront pas le placement des conteneurs en cours d'exécution, ce qui peut entraîner d'importantes variations de consommation sur les serveurs. Ces variations sans anticipation peuvent également entraîner des alertes, ce qui explique également le nombre plus important d'alertes pour les méthodes *iter-consol* et *heur*. Tout comme pour les jeux de données réels, la méthode *spread* n'entraîne pas d'alerte (sauf pour le jeu Profiles_add_5n) car elle utilise l'intégralité des serveurs disponibles dès le début, quand les autres méthodes essaient d'en utiliser moins. Pour les jeux Profiles_change_5n et Profiles_add_15n, le nombre de nœuds utilisés est le même pour toutes les méthodes alors que le nombre d'alertes est plus important : ceci s'explique par le fait que les méthodes *iter-consol*, *heur* et *loop* ont proposé un placement avec moins de nœuds en début d'exécution, mais ont été contraintes d'ouvrir un nœud supplémentaire suite à une alerte. Le nombre de nœuds finalement utilisés est donc retenu pour le critère c_1.

4.4.2 Evaluation du clustering

Les notations utilisées pour l'évaluation du clustering sont rappelées dans le Tableau 4.11.

Notation	Signification
$Silh._{mean}$	Score Silhouette moyen associé au clustering de chaque boucle
$Silh._{min}$	Score Silhouette minimum associé au clustering sur l'ensemble des boucles
$Silh._{max}$	Score Silhouette maximum associé au clustering sur l'ensemble des boucles
$moves_C$	Nombre total de changements pour le clustering
c_1	Nombre de nœuds
c_2	Amplitude maximale de consommation sur les nœuds
c_3	Variance maximale de consommation sur les nœuds
c_4	Nombre d'alertes engendrées
$moves_A$	Nombre total de déplacements pour le placement

Tableau 4.11 – Index des notations utilisées pour l'évaluation du clustering.

Les résultats concernant l'évaluation du clustering sont séparés en deux tableaux :

— un tableau pour l'évaluation intrinsèque du clustering, comprenant les scores

Silhouette moyens, minimaux et maximaux de chaque méthode, ainsi que le nombre de changements occasionnés dans le clustering ;

— un deuxième tableau présentant les résultats métier en utilisant chacune des méthodes de clustering : les critères metiers c_1, c_2, c_3 et c_4 ainsi que le nombre de déplacements engendrés pour le placement.

Le Tableau 4.12 présente les résultats liés à l'évaluation intrinsèque du clustering pour les jeux de données réels.

Dataset	Méthode	$Silh._{mean}$	$Silh._{min}$	$Silh._{max}$	$moves_C$
AlibabaV1_50n	*loop_cluster*	0,847	**0,692**	0,914	**37,01**
	kmeans_scratch	0,851	0,541	0,930	69,23
	stream_km	**0,853**	0,664	**0,935**	61,11
AlibabaV2_50n	*loop_cluster*	0,769	**0,651**	0,802	**21,54**
	kmeans_scratch	0,766	0,585	**0,810**	71,21
	stream_km	**0,771**	0,583	0,796	54,71
Alter_way_7d	*loop_cluster*	**0,952**	**0,916**	**0,964**	**18,23**
	kmeans_scratch	0,949	0,908	0,958	42,12
	stream_km	0,948	0,906	0,961	45,19

TABLEAU 4.12 – Résultats des critères intrinsèques de comparaison des méthodes de clustering pour les jeux de données réels.

En premier lieu, nous voyons que l'évaluation intrinsèque du clustering, indiquée par le score Silhouette, est très proche pour les méthodes étudiées. Ceci ne permet pas de conclure d'une meilleure efficacité pour une méthode précise, mais permet de montrer la pertinence de notre méthode de mise à jour du clustering. Sur ce tableau, la différence majeure se situe au niveau du nombre de modifications totales dans le clustering ($moves_C$) : en utilisant un nouveau *K-means* à chaque itération, nous obtenons au final plus de modifications, alors que notre boucle en entraîne moins. Cette différence s'explique par le fait que notre méthode utilise un seuil pour générer (ou non) des conflits, puis s'assure d'améliorer la qualité du clustering dans l'heuristique de modification pour effectivement modifier ce clustering.

Le Tableau 4.13 montre l'impact des différentes méthodes de clustering sur les différents indicateurs métier, sur les jeux de données réels.

Nous voyons que, malgré des résultats encore assez proches entre les trois méthodes, les critères métier sont cette fois dominés par notre méthode *loop_cluster*. De plus, sauf concernant le jeu Alter_way_7d, notre méthode occasionne une plus grande stabilité des solutions de placement (valeurs de $moves_A$ moins grandes). Une analyse détaillée des modifications du clustering en fonction des caractéristiques des données est ici difficile. C'est pourquoi nous utilisons les jeux de données synthétiques pour effectuer des analyses plus précises.

Dataset	Méthode	c_1 *nœuds*	c_2 *ampli*	c_3 *var*	c_4 *alertes*	$moves_A$
AlibabaV1_50n	*loop_cluster*	**46**	**0,00**	**0,00**	**0,20**	**12,02**
	kmeans_scratch	**46**	51,23	52,97	0,31	21,51
	stream_km	**46**	48,55	41,22	0,30	21,32
AlibabaV2_50n	*loop_cluster*	**44**	**0,00**	**0,00**	**1,4**	**23,03**
	kmeans_scratch	44,7	31,75	27,02	1,5	60,91
	stream_km	**44**	27,00	21,56	1,5	66,42
Alter_way_7d	*loop_cluster*	**2**	**0,00**	**0,00**	**0**	31,36
	kmeans_scratch	**2**	6,97	8,52	**0**	**17,01**
	stream_km	**2**	6,01	9,63	**0**	21,70

TABLEAU 4.13 – Résultats des critères métier de comparaison des méthodes de clustering pour les jeux de données réels.

Le Tableau 4.14 montre les résultats propres au clustering pour les jeux de données générés.

Dataset	Méthode	$Silh._{mean}$	$Silh._{min}$	$Silh._{max}$	$moves_C$
Profiles_stab_5n	*loop_cluster*	0,918	**0,913**	**0,924**	**0,0**
	kmeans_scratch	0,918	0,911	0,922	**0,0**
	stream_km	**0,919**	0,911	**0,924**	**0,0**
Profiles_add_5n	*loop_cluster*	0,648	0,539	**0,771**	**17,21**
	kmeans_scratch	**0,659**	**0,588**	0,752	31,41
	stream_km	0,652	0,583	0,760	24,17
Profiles_change_5n	*loop_cluster*	**0,705**	**0,661**	0,848	**11,54**
	kmeans_scratch	0,703	0,636	**0,864**	26,64
	stream_km	0,703	0,649	0,848	19,42
Profiles_stab_10n	*loop_cluster*	**0,778**	**0,692**	0,804	**3,30**
	kmeans_scratch	0,758	0,641	0,830	19,23
	stream_km	0,775	0,664	**0,835**	3,93
Profiles_add_15n	*loop_cluster*	0,569	0,503	**0,707**	**51,54**
	kmeans_scratch	**0,581**	**0,552**	0,685	52,25
	stream_km	0,577	0,583	0,796	54,71
Profiles_change_10n	*loop_cluster*	**0,640**	**0,593**	0,819	**32,81**
	kmeans_scratch	**0,640**	0,568	**0,821**	91,54
	stream_km	**0,640**	**0,593**	0,819	39,50

TABLEAU 4.14 – Résultats des critères intrinsèques de comparaison des méthodes de clustering pour les jeux de données générés.

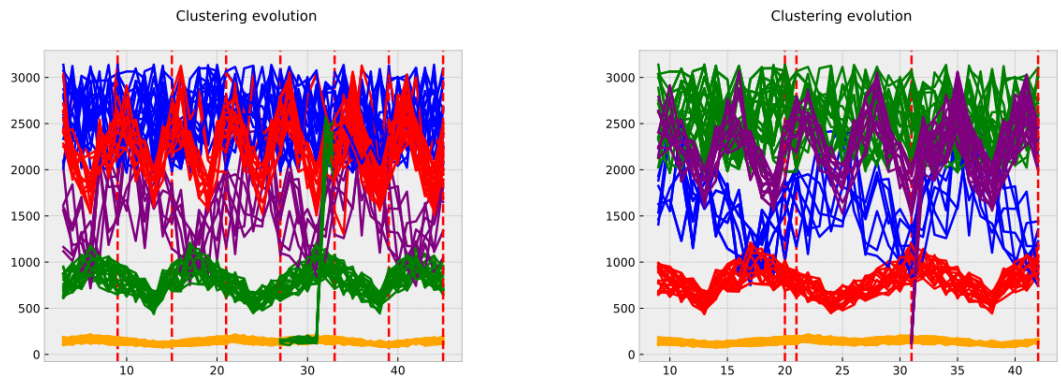
Tout comme pour les jeux de données réels, les indicateurs intrinsèques au clustering sont très proches d'une méthode à l'autre. Nous pouvons apporter une analyse supplémentaire dépendamment des caractéristiques des jeux de données créés en relation avec la valeur de τ utilisée :

— Jeux **Profiles_stab_5n** et **Profiles_stab_10n** : le premier jeu de données ayant été

créé spécifiquement comme très stable, avec des profils en opposition de phase, notre méthode (mais également les autres méthodes de mise à jour de clustering) n'entraîne aucune modification du clustering, quelle que soit la valeur de τ.

Le deuxième jeu de données, bien que stable, présente plus de profils générés aléatoirement (contrairement au jeu de données précédent). Cette caractéristique amène plus de variations dans les résultats : pour des fenêtres de temps plus courtes (par exemple $\tau = 10$), certains conteneurs sont assimilés à d'autres profils, alors que pour des fenêtres de temps plus longues (exemple avec $\tau = 20$) ces conteneurs gardent le même profil au cours de l'exécution. C'est ce qui explique les valeurs de $moves_C$ non nulles en moyenne pour ce jeu de données, malgré sa stabilité.

— Jeux **Profiles_change_5n** et **Profiles_change_10n** : dans le premier jeu, dont la création est détaillée en Section 4.3, 5 profils types différents sont créés, et 10 individus changent de profil au temps $t = 32$. Nous constatons tout d'abord que notre méthode *loop* génère en moyenne $11,54$ changements d'affectation pour le clustering, ce qui est proche des 10 individus changeant de profil. La différence s'explique par les différentes fenêtres de temps utilisées : en effet, comme l'illustre la Figure 4.17, les individus changeant de profil sont d'abord assignés au cluster *vert* en utilisant $\tau = 15$ (Figure 4.17a) avant d'être assignés, à l'itération suivante, au cluster *rouge*, alors qu'en utilisant $\tau = 25$ (Figure 4.17b) ces mêmes conteneurs sont directement assignés au cluster *violet* (étant le même que le cluster *rouge* pour $\tau = 15$). Ainsi, si nous détaillons le nombre de modifications pour le clustering sur ces valeurs de τ, nous remarquons qu'il y en 20 pour $\tau = 15$ et 10 pour $\tau = 25$. Nous notons également qu'avant le temps où les conteneurs changent effectivement de profil ($t = 32$), il n'y a aucun changement dans le clustering.

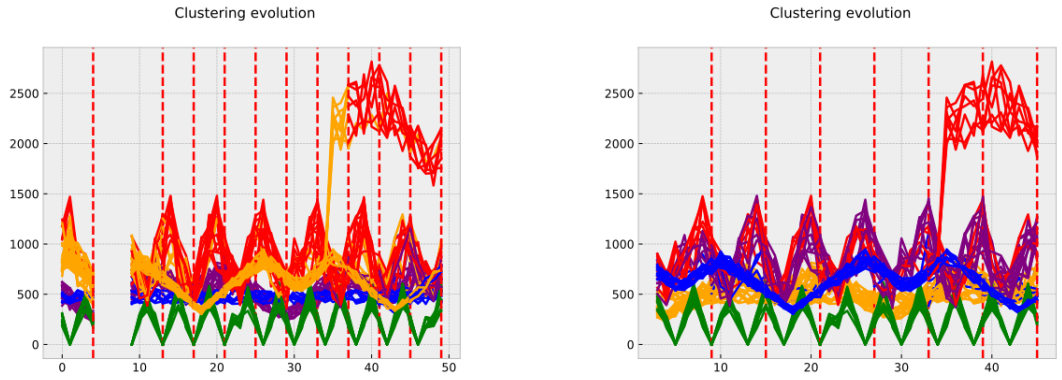


(a) $\tau = 15$ (b) $\tau = 25$

FIGURE 4.17 – Evolution des consommations de ressources des conteneurs, colorés par leur cluster, pour le jeu Profiles_change_5n.

— Jeux **Profiles_add_5n** et **Profiles_add_15n** : les observations des résultats sur les différentes valeurs de τ pour ces deux jeux de données sont similaires à celles

concernant les deux jeux de données précédents, mais avec une difficulté supplémentaire : les profils suivis par les conteneurs changeant ne sont pas a priori identifiés. Or le nombre de clusters K utilisés reste identique, il est donc plus difficile de stabiliser le clustering. C'est pour cette raison que plus de modifications du clustering interviennent pour ces jeux de données. Cette situation est illustrée en exemple en Figure 4.18 : avec $\tau = 10$ (Figure 4.18a), les individus adoptant le nouveau profil sont d'abord assigné à un cluster (*jaune*), mais dans la fenêtre suivante, ces individus étant trop éloignés de ceux déjà existants dans le cluster *jaune*, ils sont réassignés à un nouveau cluster (*rouge*). Ces réassignations nécessitent le recalcul des profils moyens, et le nombre de clusters n'étant plus adapté, de nouvelles réassignations ont lieu dans les fenêtres suivantes. En utilisant $\tau = 15$ (Figure 4.18b), les individus ayant changé de profil sont assignés au cluster *rouge*, ce qui occasionne la réassignations d'autres individus dans les fenêtres suivantes. Nous pensons qu'adapter le nombre de clusters utilisés permettrait une meilleure stabilité de ces solutions (voir Section 5.2). Cependant, les autres méthodes de mise à jour du clustering souffrent des mêmes inconvénients.



(a) $\tau = 10$ (b) $\tau = 15$

FIGURE 4.18 – Evolution des consommations de ressources des conteneurs, colorés par leur cluster, pour le jeu Profiles_add_5n.

Le Tableau 4.15 montre les résultats métier, selon les méthodes de clustering utilisées, pour les jeux de données générés.

Sur l'ensemble des critères métiers et pour tous les jeux d'essai, notre méthode *loop_cluster* domine les autres méthodes de clustering, sauf pour le jeu de données Profiles_add_5n où la méthode *stream_km* présente en moyenne un nombre d'alertes (critère c_4) moins élevé. Cette détérioration du critère c_4 est à mettre en corrélation avec le critère c_1 (nombre de nœuds), pour lequel notre boucle est la seule méthode à proposer (dans en moyenne 1 cas sur 3) une solution de placement utilisant 4 nœuds, alors que les autres méthodes en utilisent 5 dans 100% des cas.

Dataset	Méthode	c_1 *nœuds*	c_2 *ampli*	c_3 *var*	c_4 *alertes*	$moves_A$
Profiles_stab_5n	*loop_cluster*	**5**	**0,00**	**0,00**	**0**	**0,0**
	kmeans_scratch	**5**	6,17	9,51	**0**	**0,0**
	stream_km	**5**	3,23	3,12	**0**	**0,0**
Profiles_add_5n	*loop_cluster*	**4,6**	**0,00**	**0,00**	1,7	88,00
	kmeans_scratch	5	9,45	12,62	1,5	98,20
	stream_km	5	6,78	7,02	**1,4**	**82,14**
Profiles_change_5n	*loop_cluster*	**5**	**0,00**	**0,00**	**0**	**19,10**
	kmeans_scratch	**5**	28,41	19,52	**0**	44,16
	stream_km	**5**	12,28	13,93	**0**	41,70
Profiles_stab_10n	*loop_cluster*	**8,2**	**0,00**	**0,00**	**1,9**	**3,71**
	kmeans_scratch	8,8	29,33	48,11	3,8	21,51
	stream_km	8,7	21,95	29,41	4,1	4,54
Profiles_add_15n	*loop_cluster*	**15**	**0,00**	**0,00**	**1,8**	325,02
	kmeans_scratch	**15**	17,13	27,02	**1,8**	385,44
	stream_km	**15**	14,10	17,02	**1,8**	**307,07**
Profiles_change_10n	*loop_cluster*	**9,3**	**0,00**	**0,00**	**0,2**	**73,63**
	kmeans_scratch	**9,3**	38,56	33,74	0,7	165,10
	stream_km	**9,3**	25,61	26,54	0,7	103,06

TABLEAU 4.15 – Résultats des critères métiers de comparaison des méthodes de clustering pour les jeux de données générés.

La stabilité des jeux de données Profiles_stab_5n et Profiles_stab_10n, montrée précédemment avec les critères intrinsèques au clustering, se retrouve avec les critères métiers : toutes les méthodes n'entraînent aucun mouvement et des critères métiers très proches pour le premier jeu d'essai, et peu de mouvements pour le deuxième nommé malgré la plus grande variété de profils et d'individus utilisés. Par ailleurs, nous notons pour ce dernier jeu d'essai que toutes les méthodes obtiennent, après la phase d'observation T_{obs}, un placement utilisant 8 nœuds, mais la méthode *loop_cluster* réussit plus souvent (3 fois sur 4 en moyenne, contre 1 fois sur 4 et 1 fois sur 3) à maintenir ce nombre de nœuds tout le long de l'exécution au lieu d'ouvrir un nœud supplémentaire.

4.4.3 Étude de la performance

Nous résumons tout d'abord en Tableau 4.16 les notations utilisées pour décrire les résultats qui suivent. Tous les temps indiqués sont des temps *CPU*.

Les périodes T_{obs} et T_{run} sont respectivement décrites en Section 3.3 et en Section 3.4. La première boucle *pre_loop* sert d'initialisation pour les problèmes d'optimisation, en suivant les étapes suivantes :

1. les modèles $\mathcal{CCP}$ et $\mathcal{CAP}$, décrits en Section 3.2, sont construits pour la première fois ;

Notation	Signification
t_{obs}	Temps moyen de la période d'observation T_{obs} (pour calculer les solutions C_0 et A_0)
t_{pre_loop}	Temps moyen de la première boucle, dans laquelle sont construits et résolus pour la première fois les problèmes d'optimisation
t_{loop}	Temps moyen d'exécution d'une seule boucle (évaluation et mise à jour des solutions)
t_{run}	Temps total de la période T_{run} d'exécution de toutes les boucles
t_{total}	Temps total d'exécution de la méthode entière
t_{build}	Temps de construction des problèmes d'optimisation (pendant *pre_loop*)
t_{solve}	Temps de résolution des problèmes d'optimisation (pendant *pre_loop*)
t_{update}	Temps de mise à jour des problèmes d'optimisation (en moyenne sur une boucle)
$t_{resolve}$	Temps de résolution des problèmes d'optimisation (en moyenne sur une boucle)
t_{build_int}	Temps de construction des problèmes d'optimisation en nombres entiers (sans considérer la relaxation)
t_{solve_int}	Temps de résolution moyen des problèmes d'optimisation en nombres entiers (sans considérer la relaxation)

TABLEAU 4.16 – Index des notations utilisées pour l'évaluation de la performance.

2. les contraintes $mustLink_C$ et $mustLink_A$ (détaillées respectivement en Équation 3.21 et en Équation 3.24) sont ajoutées respectivement aux modèles $\mathcal{CCP}$ et $\mathcal{CAP}$;
3. les problèmes $\mathcal{CCP}$ et $\mathcal{CAP}$ sont résolus pour la première fois ;
4. les valeurs des variables duales associées aux contraintes $mustLink_C$ et $mustLink_A$ sont calculées pour la première fois (permettant ainsi leur comparaison dans les itérations suivantes).

Nous discutons ici de l'applicabilité de notre méthode en contexte réel. Comme expliqué en Section 4.2, nous relevons les temps d'exécution des différentes étapes de la méthode pour les mettre en relation avec les temps réellement écoulés dans les jeux de données. Le Tableau 4.17 montrent ces temps de calcul pour les jeux de données réels, vu qu'ils sont les seuls à correspondre à des temps *réels*.

Dans un premier temps, nous observons un temps d'exécution t_{pre_loop} de la première boucle supérieur au temps moyen t_{loop} d'exécution de toutes les autres boucles : ceci s'explique par le fait que, dans la première boucle *pre_loop*, les deux modèles d'optimisation (pour le clustering et le placement) sont construits puis résolus pour la première fois. A l'inverse, dans toutes les autres boucles, les modèles d'optimisation sont simplement mis à jour en fonction des nouvelles données, mais un grand nombre de contraintes et les variables définissant les problèmes restent identiques. De plus, leur résolution utilise la dernière résolution comme point de départ, ce qui peut réduire le temps de résolution

des problèmes.

Dataset	Temps moyens (secondes)				
	t_{obs}	t_{pre_loop}	t_{loop}	t_{run}	t_{total}
AlibabaV1_50n	27,7	76,8	37,2	240,7	268,7
AlibabaV2_50n	425,4	324,4	162,8	1165,8	1595,1
Alter_way_7d	55,1	13,2	12,3	79,2	135,6

TABLEAU 4.17 – Résultats des critères de performance pour les jeux de données réels.

Comme décrit en Section 4.3, ces jeux de données ont des correspondances de temps réel.

Concernant le jeu de données AlibabaV1_50n, les données représentent 24 heures de consommations, avec de nouvelles données remontées toutes les 10 minutes, soit toutes les 600 secondes. La durée moyenne d'une boucle pour ce jeu de données étant de 37,2 secondes, notre méthode a, de loin, le temps de mettre à jour le profilage et le placement avant l'arrivée de nouvelles données.

De la même façon, les jeux de données AlibabaV2_50n et Alter_way_7d représentent respectivement 8 jours et 7 jours de données. Concernant le jeu de données Alter_way_-7d, de part la taille du problème (seulement 3 serveurs et 354 conteneurs, contre 50 serveurs et 888 conteneurs pour AlibabaV2_50n), les temps de calcul sont très faibles, ce qui ne remet pas en cause l'utilisabilité de la méthode. Bien que les temps de calcul soient considérablement plus élevés pour le jeu AlibabaV2_50n, une boucle s'exécute en moyenne 162,8 secondes, soit moins de 3 minutes alors que de nouvelles données arrivent en moyenne toutes les 10 minutes. Donc notre méthode est bien applicable pour ce jeu de données également.

Une étude particulière a été réalisée concernant les temps d'exécution liés aux problèmes d'optimisation. Le Tableau 4.18 recense ces temps, dont les notations sont rappelées en Tableau 4.16.

Dataset	Temps moyens (secondes)					
	t_{build}	t_{solve}	t_{update}	$t_{resolve}$	t_{build_int}	t_{solve_int}
AlibabaV1_50n	53,4	7,5	6,4	4,9	76,8	24,3
AlibabaV2_50n	234,3	33,5	38,2	18,7	378,6	101,2
Alter_way_7d	9,5	0,7	3,9	0,5	12,3	0,9

TABLEAU 4.18 – Temps liés aux problèmes d'optimisation.

Tout d'abord, nous observons naturellement l'intérêt de ne pas considérer les problèmes d'optimisation à froid, en considérant la différence de temps de calcul entre la construction de ces problèmes lors de la *pre_loop*, représentée par t_{build}, et leur mise à jour dans les boucles suivantes, représentée par t_{update}.

Nous notons que les problèmes d'optimisation créés contiennent respectivement pour les jeux AlibabaV1_50n, AlibabaV2_50n et Alter_way_7d environ 330.000 variables et 90.000 contraintes, 560.000 variables et 130.000 contraintes et enfin 250.000 variables et 60.000 contraintes. Les jeux AlibabaV2_50n et Alter_way_7d présentant le même ordre de grandeur de points de données et de conteneurs, nous remarquons que le nombre de nœuds (respectivement 50 et 3) a une influence importante sur la taille des problèmes d'optimisation. Malgré la taille importante des modèles d'optimisation, leur résolution se fait en moins d'une minute, ce qui est en adéquation avec les contraintes opérationnelles du métier.

4.5 Discussion

Nous avons présenté dans un premier temps l'implémentation, le fonctionnement ainsi que l'architecture de la librairie COTS développée dans le cadre de cette thèse. Le développement ed cette librairie en suivant une architecture modulaire, laissant un grand choix de personnalisation à l'utilisateur. Cette librairie nous permet d'implémenter la méthodologie complète décrite dans le Chapitre précédent, ainsi que d'évaluer cette méthode en la comparant à d'autres méthodes de l'état de l'art.

Cette évaluation a été réalisée en détaillant chacun de ses points : l'impact sur des critères métiers identifiés, permettant ainsi l'évaluation de la performance intrinsèque de la méthode, mais également sur divers indicateurs permettant d'analyser l'impact du clustering sur le déroulement de la méthode et les résultats obtenus, ou encore d'étudier l'influence de certains paramètres de cette méthode.

Avec cette évaluation, nous montrons une amélioration des critères de performance propres au métier, en particulier par l'économie énergétique représentée par l'utilisation d'un nombre réduit de serveurs, tout en maintenant une stabilité de charge sur ces derniers, ce qui réduit les risques d'incidents de production.

Nous avons également montré que notre méthode originale de mise à jour du clustering restait cohérente du point de vue de critères intrinsèques au problème de clustering, en la comparant à d'autres méthodes de clustering de données dynamiques. De plus, les indicateurs propres au problème métier de placement, pour lequel notre modélisation du clustering a été pensée, montrent l'intérêt de cette méthode de mise à jour : les critères métiers sont améliorés avec notre méthode et les solutions obtenues sont plus stables dans le temps.

Enfin, nous avons analysé et montré l'applicabilité de notre méthode sur un environnement de production réel, en étudiant les temps nécessaires à la réalisation de chaque étape et en les comparant avec les temps réels représentés par les jeux de données. Ceci nous encourage à poursuivre les travaux d'industrialisation pour permettre cette mise

en production.

Chapitre 5

Conclusion et perspectives

Sommaire

Cette thèse s'inscrit dans le cadre de l'allocation de ressources en contexte d'hébergement en conteneurs. Afin de prendre en compte l'évolutivité des consommations dûe à l'élasticité introduite par la virtualisation en conteneurs, la stratégie de placement de ces derniers doit être ajustée dans le temps.

Dans cette optique, nous proposons un système de boucles à deux niveaux :

— Une première étape mettant à jour les profils de consommation des conteneurs, représentés par un clustering. Ce clustering est modélisé par un problème d'optimisation et les informations de la solution courante y sont ajoutées par le biais de contraintes *mustLink*. La solution de ce clustering est modifiée en utilisant un graphe de conflits, construit en utilisant les variations des valeurs des variables duales associées aux contraintes *mustLink*.

— La mise à jour du placement des conteneurs sur l'infrastructure dans un second temps, de la même manière que la mise à jour du clustering, à travers la modélisation du placement par un problème d'optimisation et l'utilisation des variables duales associées à des contraintes *mustLink* propres au placement. En revanche, l'expression de l'objectif du problème de placement utilise les profils des clusters identifiés par le clustering lors de la première étape.

Cette boucle d'exécute donc dans un contexte *online*, en continu suivant l'arrivée de nouvelles données de consommation.

Des algorithmes, exécutés dans un contexte *offline*, sont développés pour obtenir les premières solutions de clustering et de placement, qui seront ensuite mises à jour dans la boucle. Le premier clustering est obtenu en utilisant l'algorithme K-means, alors que le premier placement est obtenu grâce à une heuristique utilisant les profils des clusters.

Nous résumons dans la suite de ce chapitre les différentes contributions apportées au cours de cette thèse, ainsi que les différentes pistes d'amélioration et les travaux envisagés pour la suite de ce projet.

5.1 Contributions

Nous proposons une méthode de placement de conteneurs dans un contexte d'hébergement utilisant une méthode hybride apprentissage non supervisé - optimisation, à travers un système de boucles permettant la mise à jour du placement (et du profilage utilisé par la méthode de placement) en fonction de l'évolution de consommation des ressources. Le développement de cette méthode et son implémentation ont permis diverses contributions que l'on peut différencier selon le domaine.

5.1.1 Contribution métier

Comme il est mentionné en Section 2.1.1, les stratégies de placement de micro-services fournissent encore assez peu d'intelligence, si elles réagissent au contexte d'exécution, elles n'établissent pas un effort de profilage hors-ligne.

Dans la littérature, beaucoup de travaux se concentrent soit sur la fourniture d'un placement initial de conteneurs, sans la possibilité de les modifier dans le temps, soit sur l'ajustement des niveaux de ressources alloués aux conteneurs. Notre méthode est donc novatrice en ce qui concerne la proposition d'une stratégie de placement automatiquement ajustée dans le temps en fonction de l'évolution des consommations de ressources.

En plus de la nouveauté méthodologique, les tests réalisés (voir Section 4.2) montrent que les critères métiers identifiés sont améliorés avec notre méthode en comparaison à d'autres méthodes utilisées en production.

Par ailleurs, le travail sur l'ajustement des niveaux de ressources allouées aux conteneurs au sein de notre librairie a déjà commencé. Ceci nous permettrait de répondre à une problématique plus large concernant l'allocation de ressources (voir Section 5.2).

5.1.2 Contribution scientifique

Nous parlons ici notamment de l'utilisation combinée du clustering et de l'optimisation, encore peu mis en relation comme le montre la Section 2.2.2, pour répondre à une problématique donnée. En effet, notre méthode utilise ces deux approches en interaction pour optimiser un objectif métier, le clustering étant utilisé pour guider la solution de placement de micro-services et l'optimisation utilisée pour évaluer le clustering et le contraindre.

Par ailleurs, la méthode proposée doit répondre à des contraintes opérationnelles fortes. En effet, étant dans un contexte de *streaming* (de nouvelles métriques sont récupérées en continu), notre boucle doit fournir des solutions le plus rapidement possible. Ceci a été possible grâce à la considération des relaxations continues de plusieurs modèles d'optimisation discrète, ainsi que la définition d'objectif guidé par le clustering.

Les différents travaux ont fait l'objet de publications scientifiques lors de conférences :

— Le développement de la phase d'observation de la méthode, avec l'obtention de premiers résultats comparatifs avant l'implémentation de la boucle, a été présenté au Paris Open Source Summit (POSS) 2019 [POSS, 2019], à la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF) 2020 [ROADEF, 2020] et à International conference on Optimization and Learning (OLA) 2020 [OLA, 2020].

— L'implémentation de la boucle et les premiers résultats comparatifs de celle-ci ont été présentés à la Conférence francophone sur l'Extraction et la Gestion des Connaissances (EGC) 2022 [Leclercq et al., 2022].

5.1.3 Contribution technique

À travers cette thèse, nous avons développé une librairie Python nommée COTS (voir Section 4.1) pour implémenter notre méthode ainsi que réaliser les différentes campagnes d'évaluation. Notre volonté est de publier le code sous licence open source, non seulement car l'application dans laquelle il a été développé - le placement de ressources en contexte de conteneurs - est de plus en plus utilisée et attirerait donc une communauté intéressée par l'utilisation directe de la librairie, mais également car l'architecture de cette dernière est très modulaire. Ce dernier point permet à un utilisateur potentiel d'utiliser la librairie, en apportant son propre problème (son application spécifique), pour y répondre avec le moins de modifications possibles dans la librairie.

Par ailleurs, comme spécifié en Section 4.3, un générateur de données a été spécialement développé pour réaliser les expérimentations. Ce générateur permet de générer plusieurs types de jeux d'essai : des données complètement aléatoires, des données suivant un des profils spécifiques (que l'utilisateur peut renseigner) et en y incorporant certains changements (changement de profil de consommation, apparition de profils ...).

5.2 Perspectives

Les travaux réalisés au cours de cette thèse ont abouti à des résultats expérimentaux encourageants et validant l'intérêt de notre méthode, tant au niveau des critères métiers qu'au niveau de son utilisabilité en situation réelle. Il reste néanmoins des axes d'amélioration et des projets futurs déjà identifiés.

La première considération est de mieux prendre en compte le contexte métier réel. En effet, les tests expérimentaux ont été réalisés avec des traces de consommation historisées, et nous envisageons de les réaliser dans un cadre de production, en déployant notre méthode sur une infrastructure réelle en présence de conteneurs. Ceci nous permettrait également de mieux prendre en compte certaines contraintes métier, comme le coût de déplacement d'un conteneur, difficilement quantifiable autrement. Par ailleurs, une étude sur les stratégies de niveaux de ressources alloués aux conteneurs a déjà été entamée. Cette problématique entre dans le cadre global d'allocation de ressources et s'ajoute donc à la problématique du placement de ces conteneurs.

Comme décrit en Chapitre 3, notre méthode a de nombreux paramètres, qu'il peut être difficile de fixer. Bien que des travaux aient déjà été réalisés pour étudier ces paramètres, nous avons identifiés des pistes d'amélioration pour une meilleure gestion de certains

paramètres :

— le nombre de clusters K à utiliser : nous avons montré les différentes méthodes pour estimer le nombre optimal de clusters pour un jeu de données, mais celui-ci reste fixe au cours de la méthode. Nous envisageons donc de remettre en question ce paramètre de manière dynamique, notamment par les variables duales du problème de clustering et la génération du graphe de conflit.

— la longueur de la fenêtre temporelle τ : au cours des expérimentations menées, nous avons pu remarquer que le choix de la longueur de la fenêtre temporelle influençait la qualité du résultat. Cependant, il reste très difficile de prévoir à l'avance la taille optimale de cette fenêtre en fonction des profils à observer. C'est pourquoi nous envisageons de considérer une taille dynamique au cours de l'exécution. En fonction du nombre de déplacements observés et de la taille des graphes de conflits générés, nous pourrions baisser ou augmenter ce τ. L'application d'une règle simple n'est pas possible ici, c'est pourquoi des travaux approfondis sont prévus dans ce cadre. Nous considérons également le stockage de profils issus de différentes valeurs de τ, et se servir de ces divers profils pour le placement.

Notre méthode propose l'utilisation combinée du clustering et des techniques d'optimisation à travers l'implémentation d'une boucle. De plus, nous avons montré comment le clustering guidait l'algorithme de placement. Nous avons travaillé également sur un retour d'informations du placement pour le clustering, notamment à travers l'utilisation de contraintes *cannotLink* qui complèteraient les contraintes *mustLink* présentes. Ces travaux n'ont pour le moment pas été concluants, mais nous continuons de travailler dans ce sens pour implémenter une rétro-action du placement vers le clustering au sein de la boucle.

D'un point de vue technique, la librairie COTS a été développée dans l'optique de le partager à la communauté open source. Des travaux de factorisation, de documentation et d'amélioration globale du code ont été réalisés dans ce sens. La principale limite actuelle du code est sa dépendance au solveur CPLEX, qui est propriétaire. Le code a été retravaillé de manière à modéliser les problèmes d'optimisation sans dépendance à ce solveur spécifiquement et donc permettre l'utilisation de n'importe quel solveur, notamment un solveur open source.

Annexe A

Annexes

A.1 Paramètres de COTS

- *analysis* : liste des paramètres concernants la période d'analyse (voir Section 3.3)
  - *window_duration* : taille de la fenêtre de temps considérée
  - *sep_time* : temps séparateur des périodes d'analyse et d'optimisation
- *clustering* : paramètres concernant l'obtention du premier clustering (voir Section 3.3)
  - *algo* : choix de l'algorithme utilisé (parmi K-means, hiérarchique, spectral)
  - *nb_clusters* : nombre de clusters à utiliser
- *data* : paramètres concernant les données.
  - *individuals_file* : nom du fichier pour les consommations des conteneurs
  - *hosts_meta_file* : nom du fichier pour les informations des noeuds
  - *individual_field* : nom du champ identifiant les conteneurs dans les données
  - *host_field* : nom du champ identifiant les noeuds dans les données
  - *tick_field* : nom du champ identifiant les timestamps de données
  - *metrics* : liste des noms des ressources considérées dans les données
- *heuristic* : paramètres concernant l'heuristique de placement lors de la période d'analyse (voir Section 3.3).
  - *algo* : choix de l'heuristique utilisée pour obtenir le premier placement (parmi distant_pairwise, ffd, spread)
- *optimization* : paramètres concernant les modèles d'optimisation.
  - *model* : emplacement du fichier dans lequel les problèmes d'optimisation sont spécifiés. L'utilisateur peut utiliser ses propres modèles d'optimisation, soit pour appliquer ses propres méthodes soit pour les utiliser dans un autre contexte (voir Section pour plus de détails).

  - *solver* : choix du solver utilisé
- *loop* : paramètres concernant les boucles.
  - *mode* : mode de déclenchement des boucles (parmi event, sequential, hybrid)
  - *tick* : le nombre de points de données utilisés pour progresser dans le temps avant de déclencher une boucle (pour les modes sequential et hybrid)
  - *constraints_dual* : liste des types de contraintes utilisées pour la comparaison des variables duales lors de l'évaluation des solutions
  - *tol_dual_clust* : seuil de tolérance pour la comparaison des variables duales lors de l'évaluation du clustering
  - *tol_move_clust* : nombre de déplacements maximum autorisés lors de l'évaluation du clustering
  - *tol_dual_place* : seuil de tolérance pour la comparaison des variables duales lors de l'évaluation du placement
  - *tol_move_place* : : nombre de déplacements maximum autorisés lors de l'évaluation du placement
  - *tol_step* : facteur d'incrémentation des tolérances à chaque boucle
- *plot* : fixe les paramètres d'affichage des graphiques
  - *renderer* : choix de la méthode de rendu utilisé par matplotlib
- *allocation* : paramètres concernant l'ajustement des quantités de ressources allouées aux conteneurs
  - *enable* : active ou désactive l'ajustement dynamique des quantités de ressources
  - *constraints* : fixe les contraintes utilisées pour l'ajustement dynamique de ressources
    - *load_threshold* : seuil de remplissage maximum des noeuds
    - *max_amplitude* : amplitude maximale de consommation des noeuds
  - *objective* : fixe les objectifs recherchés pour l'ajustement dynamique des ressources
    - *open_nodes* : nombre noeuds utilisés
    - *target_load_CPU* : taux de remplissage des noeuds (pour le CPU)
- *placement* : fixe les paramètres pour le problème de placement de conteneurs
  - *enable* : active ou désactive le problème de placement de conteneurs

Liste des figures

Liste des tableaux

Liste des acronymes

ACO Algorithme de Colonies de Fourmis (Ant Colony Optimization). 21

COTS Continuous Optimization for Time Series. 1, 18, 77–83, 115, 120, 121

CPLEX IBM ILOG CPLEX Optimization Studio. 81, 121

DNN Réseaux de neuronnes profonds (Deep Neural Network). 14

DTW Déformation dynamique temporelle (Dynamic Time Warping). 31

ED Distance Euclidienne (Euclidean Distance). 22

EGC Conférence francophone sur l'Extraction et la Gestion des Connaissances. 120

FaaS Function as a Service. 15

GLPK GNU Linear Programming Kit. 81

IaaS Infrastructure as a Service. 9

LIPN Laboratoire d'Informatique Paris Nord. 12

LXC LinuX Container. 10

NIST Institut National des Standards et de la Technologie (National Institute of Standards and Technology). 8

OLA International conference on Optimization and Learning. 119

OOME Erreur de mémoire insuffisante (Out Of Memory Exception). 84

OS Système d'exploitation (Operating System). 10

PaaS Platform as a Service. 9

PLNE Programme Linéaire en Nombres Entiers. 36, 38, 39, 72

POSS Paris Open Source Summit. 119

RD Recherche et Développement. 12

ROADEF Société Française de Recherche Opérationnelle et d'Aide à la Décision. 119

SaaS Software as a Service. 9

VM Machine Virtuelle. 10

Glossaire

DevOps Pratique visant à l'unification du développement logiciel (Dev) et de l'administration des infrastructures informatiques (Ops). 12

K-means Algorithme utilisé pour calculer un clustering. 24, 41, 73, 80, 108

Python Langage de programmation. 18, 78

autoscalers Méthode ajustant dynamiquement le niveau de ressources allouées au sein d'une infrastructure. 21

bare metal Matériel informatique utilisé sans système d'exploitation. 10

branch-and-bound Méthode générique de résolution de problèmes d'optimisation combinatoire. 14

cloud computing Modèle regroupant des technologies permettant l'accès à un réseau à la demande et à un ensemble partagé de ressources informatiques configurables. 1, 8

from scratch En partant de rien. 88

génération de colonnes Méthode pour résoudre efficacement les problèmes d'optimisation linéaire de grande taille et qui consiste à décomposer l'ensemble des contraintes en deux sous-ensembles. 14

offline Hors ligne. 1, 35, 89, 118

offloading Déchargement. 12

online En ligne. 1, 35, 44, 118

overhead Surcharge. 12

packager Regrouper un ensemble de fichiers, librairies ou autres dépendances en une entité (généralement une application). 10

preprocessing Prétraitement nécessaire au début de l'exécution d'un processus. 80

runtime Temps d'exécution. 10

streaming Diffusion en continu. 16, 34, 41, 42, 119

dendrogramme Diagramme en arborescence fréquemment utilisé pour représenter un partitionnement hiérarchique. 26

NP-difficile Problème dont on ne connait pas d'algorithme permettant de le résoudre en temps polynomial (Non-deterministic Polynomial-time hardness). 24

solver Logiciel permettant de résoudre des problèmes mathématiques. 81

élasticité Capacité d'un système à ajouter et à supprimer des ressources, ou modifier leur configuration, pour s'adapter à la variation de charge en temps réel. 1, 11, 15, 20, 21, 118

Bibliographie

[Ackermann et al., 2012] Ackermann, M. R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., and Sohler, C. (2012). Streamkm++ : A clustering algorithm for data streams. *ACM J. Exp. Algorithmics*, 17. 37, 90

[Aggarwal, 2018] Aggarwal, C. (2018). A survey of stream clustering algorithms. *Data Clustering : Algorithms and Applications*, pages 231–258. 36

[Aggarwal, 2005] Aggarwal, C. C. (2005). On change diagnosis in evolving data streams. *IEEE Transactions on Knowledge and Data Engineering*, 17(5) :587–600. 36

[Aggarwal et al., 2003a] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2003a). A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, page 81–92. VLDB Endowment. 37

[Aggarwal et al., 2003b] Aggarwal, C. C., Philip, S. Y., Han, J., and Wang, J. (2003b). A framework for clustering evolving data streams. In *Proceedings 2003 VLDB conference*, pages 81–92. Elsevier. 37

[Aghabozorgi et al., 2015] Aghabozorgi, S., Seyed Shirkhorshidi, A., and Ying Wah, T. (2015). Time-series clustering – a decade review. *Information Systems*, 53 :16–38. 34

[Ágoston and E-Nagy, 2021] Ágoston, K. C. and E-Nagy, M. (2021). Mixed integer linear programming formulation for k-means cluster problem. . 41

[Agrawal et al., 1993] Agrawal, R., Faloutsos, C., and Swami, A. (1993). Efficient similarity search in sequence databases. *Lecture Notes in Computer Science*, 730 :69–84. 33

[Agrawal et al., 1998] Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 94–105. 31

[Ailon et al., 2009] Ailon, N., Jaiswal, R., and Monteleoni, C. (2009). Streaming k-means approximation. In *Advances in Neural Information Processing Systems 22 - Proceedings of the 2009 Conference*, pages 10–18. 37

[Al-Dhuraibi et al., 2017] Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., and Merle, P. (2017). Elasticity in cloud computing : state of the art and research challenges. *IEEE Transactions on Services Computing*, 11(2) :430–447. 13, 22

[Alibaba, 2017a] Alibaba (2017a). Alibaba cluster trace program. Available at `https://github.com/alibaba/clusterdata`. 94

[Alibaba, 2017b] Alibaba (2017b). Alibaba cluster trace program - v2017. Available at `https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2017`. 95

[Alibaba, 2018] Alibaba (2018). Alibaba cluster trace program - v2018. Available at `https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018`. 95

[Aloise et al., 2012] Aloise, D., Hansen, P., and Liberti, L. (2012). An improved column generation algorithm for minimum sum-of-squares clustering. *Math. Program.*, 131 :195–220. 40

[Alter way, 2017] Alter way (2017). Wolphin project. `http://wolphin.gitlab.io/`. 14

[Alter way, 2021] Alter way (2021). Alter way's website. `https://www.alterway.fr/`. 14

[Altintas et al., 2005] Altintas, Y., Brecher, C., Weck, M., and Witt, S. (2005). Virtual machine tool. *CIRP annals*, 54(2) :115–138. 12

[Ambi Karthikeyan, 2018] Ambi Karthikeyan, S. (2018). Introduction to azure iaas. In *Practical Microsoft Azure IaaS*, pages 1–38. Springer. 11

[Amini et al., 2014] Amini, A., Wah, T., and Saboohi, H. (2014). On density-based data streams clustering algorithms : A survey. *Journal of Computer Science and Technology*, 29 :116–141. 37

[Andrychowicz et al., 2016] Andrychowicz, M., Denil, M., Gómez, S., Hoffman, M., Pfau, D., Schaul, T., and Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems*, pages 3981–3989. 16

[Arthur and Vassilvitskii, 2006] Arthur, D. and Vassilvitskii, S. (2006). k-means++ : The advantages of careful seeding. Technical report, Stanford. 27, 37, 58, 91

[Awada and Barker, 2017] Awada, U. and Barker, A. (2017). Resource efficiency in container-instance clusters. *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing*. 23

[Babaki et al., 2014] Babaki, B., Guns, T., and Nijssen, S. (2014). Constrained clustering using column generation. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 438–454. Springer. 40

[Baragona, 2001] Baragona, R. (2001). A simulation study on clustering time series with meta-heuristic methods. *Quad. Stat.*, 3 :1–26. 33

[Barna et al., 2017] Barna, C., Khazaei, H., Fokaefs, M., and Litoiu, M. (2017). Delivering elastic containerized cloud applications to enable devops. *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 65–75. 23

[Basu et al., 2008] Basu, S., Davidson, I., and Wagstaff, K. (2008). *Constrained Clustering : Advances in Algorithms, Theory, and Applications*. Chapman & Hall/CRC, 1 edition. 40

[Bengio et al., 2021] Bengio, Y., Lodi, A., and Prouvost, A. (2021). Machine learning for combinatorial optimization : A methodological tour d'horizon. *European Journal of Operational Research*, 290(2) :405–421. 16

[Berkhin, 2006] Berkhin, P. (2006). A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer. 25

[Bernstein, 2014] Bernstein, D. (2014). Containers and cloud : From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3) :81–84. 12

[Bertsimas and Dunn, 2019] Bertsimas, D. and Dunn, J. (2019). *Machine learning under a modern optimization lens*. Dynamic Ideas LLC. 16

[Bhardwaj and Krishna, 2021] Bhardwaj, A. and Krishna, C. R. (2021). Virtualization in cloud computing : Moving from hypervisor to containerization - a survey. *Arabian Journal of Science and Engineering*, 46(9) :8585. 14

[Blömer et al., 2016] Blömer, J., Lammersen, C., Schmidt, M., and Sohler, C. (2016). Theoretical analysis of the k-means algorithm–a survey. In *Algorithm Engineering*, pages 81–116. Springer. 27

[Bradley et al., 2002] Bradley, P., Gehrke, J., Ramakrishnan, R., and Srikant, R. (2002). Scaling mining algorithms to large databases. *Communications of the ACM*, 45(8) :38–43. 34

[Bramer, 2007] Bramer, M. (2007). *Clustering*, pages 221–238. Springer London, London. 24

[Bubeck, 2015] Bubeck, S. (2015). Convex optimization : Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4) :231–357. 15

[Caliński and Harabasz, 1974] Caliński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1) :1–27. 27

[Cao et al., 2006] Cao, F., Ester, M., Qian, W., and Zhou, A. (2006). Density-based clustering over an evolving data stream with noise. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, volume 2006. 37

[Casalicchio, 2016] Casalicchio, E. (2016). Autonomic orchestration of containers : Problem definition and research challenges. In *Proceedings of the 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 287–290. 23

[Celebi et al., 2013] Celebi, M. E., Kingravi, H. A., and Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert systems with applications*, 40(1) :200–210. 27

[Chen and Tu, 2007] Chen, Y. and Tu, L. (2007). Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, page 133–142, New York, NY, USA. Association for Computing Machinery. 37

[Cheng et al., 2018] Cheng, W., Wang, W., and Batista, S. (2018). Grid-based clustering. In *Data clustering*, pages 128–148. Chapman and Hall/CRC. 31

[Chung and Graham, 1997] Chung, F. R. and Graham, F. C. (1997). *Spectral graph theory*, volume 92. American Mathematical Soc. 31

[Corne et al., 2012] Corne, D., Dhaenens, C., and Jourdan, L. (2012). Synergies between operations research and data mining : The emerging use of multi-objective approaches. *European Journal of Operational Research*, 221 :469–479. 16

[Cornuéjols and Miclet, 2011] Cornuéjols, A. and Miclet, L. (2011). *Apprentissage artificiel : concepts et algorithmes*. Editions Eyrolles. 38

[Dao et al., 2018] Dao, T.-B.-H., Kuo, C.-T., Ravi, S. S., Vrain, C., and Davidson, I. (2018). Descriptive clustering : Ilp and cp formulations with applications. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1263–1269. International Joint Conferences on Artificial Intelligence Organization. 40

[Deshpande et al., 2014] Deshpande, P., Sharma, S. C., and Peddoju, S. K. (2014). Implementation of a private cloud : a case study. In *Proceedings of the Third International Conference on Soft Computing for Problem Solving*, pages 635–647. Springer. 11

[Ding et al., 2002] Ding, C., He, X., Zha, H., and Simon, H. (2002). Adaptive dimension reduction for clustering high dimensional data. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 147–154. 15

[Ding et al., 2008] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. (2008). Querying and mining of time series data : Experimental comparison of representations and distance measures. *PVLDB*, 1 :1542–1552. 33

[Docker, 2013] Docker (2013). Use containers to build, share and run your applications. `https://www.docker.com/resources/what-container`. 12

[Docker, 2014a] Docker (2014a). Deploy services to a swarm. `https://docs.docker.com/engine/swarm/services/`. 86

[Docker, 2014b] Docker (2014b). Runtime options with memory, cpus, and gpus. `https://docs.docker.com/config/containers/resource_constraints/`. 86

[Docker, 2016] Docker (2016). Docker swarm strategies. `https://dker.ru/docs/component-projects/docker-swarm/scheduling/strategies/`. 22

[Docker, 2017a] Docker (2017a). Developers bring their ideas to life with docker. `https://www.docker.com/why-docker/`. 13

[Docker, 2017b] Docker (2017b). Docker : swarm mode overview. `https://docs.docker.com/engine/swarm/`. 13

[Dolan and More, 2002] Dolan, E. D. and More, J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91 :201–213. 102

[Duong et al., 2017] Duong, K.-C., Vrain, C., et al. (2017). Constrained clustering by constraint programming. *Artificial Intelligence*, 244 :70–94. 40

[Ertöz et al., 2003] Ertöz, L., Steinbach, M., and Kumar, V. (2003). Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM international conference on data mining*, pages 47–58. SIAM. 30

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231. 30

[Fatemi Moghaddam et al., 2015] Fatemi Moghaddam, F., Ahmadi, M., Sarvari, S., Eslami, M., and Golkar, A. (2015). Cloud computing challenges and opportunities : A survey. In *2015 1st International Conference on Telematics and Future Generation Networks (TAFGEN)*, pages 34–38. 14

[Fisher, 1996] Fisher, D. (1996). Iterative optimization and simplification of hierarchical clusterings. *Journal of artificial intelligence research*, 4 :147–178. 30

[Fokaefs et al., 2016] Fokaefs, M., Barna, C., Veleda, R., Litoiu, M., Wigglesworth, J., and Mateescu, R. (2016). Enabling devops for containerized data-intensive applications : An exploratory study. *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*, pages 138–148. 23

[Fortet, 1960] Fortet, R. (1960). Applications de l'algebre de boole en recherche opérationelle. *Revue Française de Recherche Opérationelle*, 4(14) :17–26. 62

[Fox et al., 2017] Fox, G. C., Ishakian, V., Muthusamy, V., and Slominski, A. (2017). Status of serverless computing and function-as-a-service (faas) in industry and research. *arXiv preprint arXiv :1708.08028*. 17

[Geelan et al., 2009] Geelan, J. et al. (2009). Twenty-one experts define cloud computing. *Cloud Computing Journal*, 4 :1–5. 10

[Geoffrion, 1974] Geoffrion, A. (1974). Lagrangian relaxation and its uses in integer programming mathematical programming study. *Math. Prog. Study*, 2 :82–114. 39

[Ghesmoune et al., 2016] Ghesmoune, M., Lebbah, M., and Azzag, H. (2016). State-of-the-art on clustering data streams. *Big Data Analytics*, 1 :13. 37

[Gilpin and Davidson, 2017] Gilpin, S. and Davidson, I. (2017). A flexible ilp formulation for hierarchical clustering. *Artificial Intelligence*, 244 :95–109. Combining Constraint Solving with Mining and Learning. 41

[Gilpin et al., 2013] Gilpin, S., Nijssen, S., and Davidson, I. (2013). Formalizing hierarchical clustering as integer linear programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 27(1) :372–378. 41

[Golay et al., 1998] Golay, X., Kollias, S., Stoll, G., Meier, D., Valavanis, A., and Boesiger, P. (1998). A new correlation-based fuzzy logic clustering algorithm for fmri. *Magnetic resonance in medicine*, 40(2) :249–260. 25

[Grabusts and Borisov, 2002] Grabusts, P. and Borisov, A. (2002). Using grid-clustering methods in data classification. In *Proceedings. International Conference on Parallel Computing in Electrical Engineering*, pages 425–426. IEEE. 31

[Gulati et al., 2015] Gulati, H., Singh, P., et al. (2015). Clustering techniques in data mining : A comparison. In *2015 2nd international conference on computing for sustainable global development (INDIACom)*, pages 410–415. IEEE. 35

[Gunopulos and Das, 2000] Gunopulos, D. and Das, G. (2000). Time series similarity measures (tutorial pm-2). In *Tutorial notes of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–307. 32

[Hassen, 2016] Hassen, A. (2016). *A survey of docker swarm scheduling strategies*. 23

[Hautamaki et al., 2008] Hautamaki, V., Nykanen, P., and Franti, P. (2008). Time-series clustering by approximate prototypes. In *2008 19th International conference on pattern recognition*, pages 1–4. IEEE. 34

[Inc., 2013] Inc., R. H. (2013). What is virtualization ? `https://opensource.com/resources/virtualization`. 12

[J. Reddi et al., 2018] J. Reddi, S., Kale, S., and Kumar, S. (2018). On the convergence of adam and beyond. In *International Conference on Learning Representations*, pages 1–23. 16

[Jain et al., 1999] Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering : a review. *ACM computing surveys (CSUR)*, 31(3) :264–323. 25

[Jarman, 2020] Jarman, A. M. (2020). Hierarchical cluster analysis : Comparison of single linkage, complete linkage, average linkage and centroid linkage method. *Georgia Southern University*. 29

[JingMao and YanXia, 2015] JingMao, Z. and YanXia, S. (2015). Review on spectral methods for clustering. In *2015 34th Chinese Control Conference (CCC)*, pages 3791–3796. 31

[Jordan and Mitchell, 2015] Jordan, M. I. and Mitchell, T. M. (2015). Machine learning : Trends, perspectives, and prospects. *Science*, 349(6245) :255–260. 15

[Jourdan et al., 2006] Jourdan, L., Dhaenens, C., and Talbi, E.-G. (2006). Using datamining techniques to help metaheuristics : A short survey. In *Lecture Notes in Computer Science*. 16

[Kaewkasi and Chuenmuneewong, 2017] Kaewkasi, C. and Chuenmuneewong, K. (2017). Improvement of container scheduling for docker using ant colony optimization. *2017 9th International Conference on Knowledge and Smart Technology (KST)*, pages 254–259. 23

[Kalpakis et al., 2001] Kalpakis, K., Gada, D., and Puttagunta, V. (2001). Distance measures for effective clustering of arima time-series. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 273–280. 33

[Kaufman and Rousseeuw, 1990] Kaufman, L. and Rousseeuw, P. J. (1990). Partitioning around medoids (program pam). *Finding groups in data : an introduction to cluster analysis*, 344 :68–125. 27, 34

[Keogh and Pazzani, 1998] Keogh, E. J. and Pazzani, M. J. (1998). An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Kdd*, volume 98, pages 239–243. 34

[Keskar and Socher, 2017] Keskar, N. and Socher, R. (2017). Improving generalization performance by switching from adam to sgd. *ArXiv*. 16

[Khalilian, 2010] Khalilian, M. (2010). Data stream clustering : Challenges and issues. *Lecture Notes in Engineering and Computer Science*, 2180. 36

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics. 15

[King, 1967] King, B. (1967). Step-wise clustering procedures. *Journal of the American Statistical Association*, 62(317) :86–101. 29

[Kokate et al., 2018] Kokate, U., Deshpande, A., Mahalle, P., and Patil, P. (2018). Data stream clustering techniques, applications, and models : comparative analysis and discussion. *Big Data and Cognitive Computing*, 2(4) :32. 37

[Kranen et al., 2011] Kranen, P., Assent, I., Baldauf, C., and Seidl, T. (2011). The clustree : Indexing micro-clusters for anytime stream mining. *Knowl. Inf. Syst.*, 29 :249–272. 37

[Kriegel et al., 2011] Kriegel, H.-P., Kröger, P., Sander, J., and Zimek, A. (2011). Density-based clustering. *Wiley interdisciplinary reviews : data mining and knowledge discovery*, 1(3) :231–240. 30

[Kubernetes, 2018] Kubernetes (2018). Kubernetes. `https://kubernetes.io/`. 13

[Kumar et al., 2002] Kumar, M., Patel, N. R., and Woo, J. (2002). Clustering seasonality patterns in the presence of errors. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, page 557–563, New York, NY, USA. Association for Computing Machinery. 33

[Leclercq et al., 2022] Leclercq, E., Rivalan, J., Roupin, F., and Rouveirol, C. (2022). Allocation de ressources par une méthode hybride machine learning-optimisation en contexte de conteneurs. In *EGC*, pages 499–500. 122

[Li et al., 2010] Li, Y., Li, W., and Jiang, C. (2010). A survey of virtual machine system : Current technology and future trends. In *2010 Third International Symposium on Electronic Commerce and Security*, pages 332–336. IEEE. 12

[Liberti et al., 2014] Liberti, L., Lavor, C., Maculan, N., and Mucherino, A. (2014). Euclidean distance geometry and applications. *SIAM review*, 56(1) :3–69. 24

[Lloyd, 1982] Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2) :129–137. 26

[Lodi and Zarpellon, 2017] Lodi, A. and Zarpellon, G. (2017). On learning and branching : a survey. *Top*, 25(2) :207–236. 16

[MacQueen, 1967a] MacQueen, J. (1967a). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium On Mathematical Statistics and Probabilities*. 26, 27

[MacQueen, 1967b] MacQueen, J. (1967b). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium On Mathematical Statistics and Probabilities*. 34, 57

[Mattetti et al., 2015] Mattetti, M., Shulman-Peleg, A., Allouche, Y., Corradi, A., Dolev, S., and Foschini, L. (2015). Securing the infrastructure and the workloads of linux containers. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 559–567. IEEE. 13

[McQuitty, 1957] McQuitty, L. L. (1957). Elementary linkage analysis for isolating orthogonal and oblique types and typal relevancies. *Educational and psychological measurement*, 17(2) :207–229. 29

[Meesrikamolkul et al., 2012] Meesrikamolkul, W., Niennattrakul, V., and Ratanamahatana, C. A. (2012). Shape-based clustering for time series data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 530–541. Springer. 33

[Meesuksabai et al., 2011] Meesuksabai, W., Kangkachit, T., and Waiyamai, K. (2011). Hue-stream : Evolution-based clustering technique for heterogeneous data streams

with uncertainty. In *Proceedings of the 7th International Conference on Advanced Data Mining and Applications - Volume Part II*, ADMA'11, page 27–40, Berlin, Heidelberg. Springer-Verlag. 37

[Mell et al., 2011] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing. *Computer Security Division, Information Technology Laboratory, National ...*. 10, 11

[Mitsa, 2010] Mitsa, T. (2010). *Temporal data mining*. Chapman and Hall/CRC. 33

[Mohajer et al., 2011] Mohajer, M., Englmeier, K.-H., and Schmid, V. J. (2011). A comparison of gap statistic definitions with and without logarithm function. 89

[Mohar et al., 1991] Mohar, B., Alavi, Y., Chartrand, G., and Oellermann, O. (1991). The laplacian spectrum of graphs. *Graph theory, combinatorics, and applications*, 2(871-898) :12. 32

[Morabit et al., 2021] Morabit, M., Desaulniers, G., and Lodi, A. (2021). Machine-learning–based column selection for column generation. *Transportation Science*, 55(4) :815–831. 16

[Mousavi et al., 2015] Mousavi, M., Bakar, A., and Vakilian, M. (2015). Data stream clustering algorithms : A review. *SOCO 2015*, 7 :1–15. 36

[Mquantin, 2017] Mquantin (2017). Illustration de l'algorithme k-means pour k=4. `https://commons.wikimedia.org/w/index.php?curid=61321400`. 27, 127

[Mueller and Kramer, 2010] Mueller, M. and Kramer, S. (2010). Integer linear programming models for constrained clustering. In Pfahringer, B., Holmes, G., and Hoffmann, A., editors, *Discovery Science*, pages 159–173, Berlin, Heidelberg. Springer Berlin Heidelberg. 40, 41

[Murtagh and Contreras, 2011] Murtagh, F. and Contreras, P. (2011). Algorithms for hierarchical clustering : an overview. *Data Mining and Knowledge Discovery*, 2 :86–97. 28

[Murtagh and Contreras, 2017] Murtagh, F. and Contreras, P. (2017). Algorithms for hierarchical clustering : an overview, ii. *Data Mining and Knowledge Discovery*, 7. 28

[N. Vhatkar and Bhole, 2019] N. Vhatkar, K. and Bhole, G. P. (2019). Optimal container resource allocation in cloud architecture : A new hybrid model. *Journal of King Saud University - Computer and Information Sciences*. 54

[Nguyen et al., 2015] Nguyen, H.-L., Woon, Y.-K., and Ng, W.-K. (2015). A survey on data stream clustering and classification. *Knowl. Inf. Syst.*, 45(3) :535–569. 36

[Nielsen, 2016] Nielsen, F. (2016). *Hierarchical Clustering*, pages 195–211. Springer International Publishing, Cham. 28

[Niennattrakul and Ratanamahatana, 2007] Niennattrakul, V. and Ratanamahatana, C. A. (2007). Inaccuracies of shape averaging method using dynamic time warping

for time series data. In *International conference on computational science*, pages 513–520. Springer. 34

[OLA, 2020] OLA (2020). International conference on optimization and learning (ola2020). Available at `https://ola2020.sciencesconf.org/`. 121

[Ouali et al., 2016a] Ouali, A., Loudni, S., Lebbah, Y., Boizumault, P., and Zimmermann, A. (2016a). Efficiently finding conceptual clustering models with integer linear programming. In *25th International Joint Conferences on Artificial Intelligence*. 15

[Ouali et al., 2016b] Ouali, A., Loudni, S., Lebbah, Y., Boizumault, P., and Zimmermann, A. (2016b). Efficiently finding conceptual clustering models with integer linear programming. In *25th International Joint Conferences on Artificial Intelligence*. 40, 41

[Pahl, 2015] Pahl, C. (2015). Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3) :24–31. 11

[Pandit et al., 2011] Pandit, S., Gupta, S., et al. (2011). A comparative study on distance measuring approaches for clustering. *International Journal of Research in Computer Science*, 2(1) :29–31. 24

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830. 57

[POSS, 2019] POSS (2019). Paris open source summit 2019. Available at `https://2019.opensourcesummit.paris/`. 121

[Prakash et al., 2018] Prakash, C., Prashanth, P., Bellur, U., and Kulkarni, P. (2018). Deterministic container resource management in derivative clouds. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 79–89. IEEE. 13

[Pyomo, 2020] Pyomo (2020). Pyomo. Available at `http://www.pyomo.org/`. 83

[Rai and Singh, 2010] Rai, P. and Singh, S. (2010). A survey of clustering techniques. *International Journal of Computer Applications*, 7(12) :1–5. 25

[Rani and Ranjan, 2014] Rani, D. and Ranjan, R. K. (2014). A comparative study of saas, paas and iaas in cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(6). 11

[Rani and Sikka, 2012] Rani, S. and Sikka, G. (2012). Article : Recent techniques of clustering of time series data : A survey. *International Journal of Computer Applications*, 52(15) :1–9. Full text available. 32

[Ren et al., 2012] Ren, K., Wang, C., and Wang, Q. (2012). Security challenges for the public cloud. *IEEE Internet computing*, 16(1) :69–73. 11

[Rezvani et al., 2014] Rezvani, M., Akbari, M. K., and Javadi, B. (2014). Resource Allocation in Cloud Computing Environments Based on Integer Linear Programming. *The Computer Journal*, 58(2) :300–314. 23, 54

[ROADEF, 2020] ROADEF (2020). 21ème congrès annuel de la société française de recherche opérationnelle et d'aide à la décision (roadef). Available at `https://roadef2020.sciencesconf.org/`. 121

[Rossi et al., 2008] Rossi, F., van Beek, P., and Walsh, T. (2008). Chapter 4 constraint programming. In van Harmelen, F., Lifschitz, V., and Porter, B., editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 181–211. Elsevier. 40

[Rousseeuw, 1987] Rousseeuw, P. J. (1987). Silhouettes : A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20 :53–65. 91

[Schölkopf et al., 1998] Schölkopf, B., Smola, A., and Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5) :1299–1319. 27

[Schütze et al., 2008] Schütze, H., Manning, C. D., and Raghavan, P. (2008). *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge. 26

[Sdo, 2018] Sdo (2018). Représentation graphique de la relaxation continue d'un problème. `https://commons.wikimedia.org/w/index.php?curid=1051376`. 39, 127

[Selim and Ismail, 1984] Selim, S. Z. and Ismail, M. A. (1984). K-means-type algorithms : A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on pattern analysis and machine intelligence*, 1 :81–87. 26

[Selvi et al., 2014] Selvi, S., Valliyammai, C., and Dhatchayani, V. N. (2014). Resource allocation issues and challenges in cloud computing. In *2014 International Conference on Recent Trends in Information Technology*, pages 1–6. 14

[Senin, 2008] Senin, P. (2008). Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 855(1-23) :40. 33

[Sharma et al., 2016] Sharma, P., Chaufournier, L., Shenoy, P., and Tay, Y. (2016). Containers and virtual machines at scale : A comparative study. In *Proceedings of the 17th international middleware conference*, pages 1–13. 13

[Shi and Malik, 2000] Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8) :888–905. 31

[Silva et al., 2014] Silva, J., Faria, E., Barros, R., Hruschka, E., de Carvalho, A., and Gama, J. (2014). Data stream clustering : A survey. *ACM Computing Surveys*, 46. 36

[SMILE, 2021] SMILE (2021). Smile's website. `https://www.smile.eu/fr`. 15

[Sokal, 1958] Sokal, R. R. (1958). A statistical method for evaluating systematic relationships. *Univ. Kansas, Sci. Bull.*, 38 :1409–1438. 29

[Song et al., 2019] Song, H., Triguero, I., and Özcan, E. (2019). A review on the self and dual interactions between machine learning and optimisation. *Progress in Artificial Intelligence*, 8(2) :143–165. 15

[Srinivasan et al., 2015] Srinivasan, A., Quadir, M. A., and Vijayakumar, V. (2015). Era of cloud computing : A new insight to hybrid cloud. *Procedia Computer Science*, 50 :42–51. 11

[Stavrinides and Karatza, 2016] Stavrinides, G. L. and Karatza, H. D. (2016). Scheduling different types of applications in a saas cloud. In *Proceedings of the 6th International Symposium on Business Modeling and Software Design (BMSD'16)*, pages 144–151. 11

[Sun et al., 2019] Sun, S., Cao, Z., Zhu, H., and Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. 15

[Sureshkumar and Rajesh, 2017] Sureshkumar, M. and Rajesh, P. (2017). Optimizing the docker container usage based on load scheduling. In *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, pages 165–168. 23

[Swarndeep Saket and Pandya, 2016] Swarndeep Saket, J. and Pandya, S. (2016). An overview of partitioning algorithms in clustering techniques. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 5(6) :1943–1946. 26

[Tang et al., 2020] Tang, W., Yang, Y., Zeng, L., and Zhan, Y. (2020). Size constrained clustering with milp formulation. *IEEE Access*, 8 :1587–1599. 40

[Tibshirani et al., 2001] Tibshirani, R., Walther, G., and Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society : Series B (Statistical Methodology)*, 63(2) :411–423. 27, 89

[Tind and Wolsey, 1981] Tind, J. and Wolsey, L. A. (1981). An elementary survey of general duality theory in mathematical programming. *Mathematical Programming*, 21 :241–261. 39

[Tosatto et al., 2015] Tosatto, A., Ruiu, P., and Attanasio, A. (2015). Container-based orchestration in cloud : State of the art and challenges. *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 70–75. 15, 23

[Udommanetanakit et al., 2007] Udommanetanakit, K., Rakthanmanon, T., and Waiyamai, K. (2007). E-stream : Evolution-based technique for stream clustering. In Alhajj, R., Gao, H., Li, J., Li, X., and Zaïane, O. R., editors, *Advanced Data Mining and Applications*, pages 605–615, Berlin, Heidelberg. Springer Berlin Heidelberg. 37

[Usmani and Singh, 2016] Usmani, Z. and Singh, S. (2016). A survey of virtual machine placement techniques in a cloud data center. *Procedia Computer Science*, 78 :491–498. 1st International Conference on Information Security and Privacy 2015. 12, 15, 22

[Vaquero et al., 2008] Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2008). A break in the clouds : towards a cloud definition. *ACM sigcomm computer communication review*, 39(1) :50–55. 10

[Vlachos et al., 2004] Vlachos, M., Gunopulos, D., and Das, G. (2004). Indexing time-series under conditions of noise. In *Data mining in time series databases*, pages 67–100. World Scientific. 33

[Vlachos et al., 2003] Vlachos, M., Lin, J., Keogh, E., and Gunopulos, D. (2003). A wavelet-based anytime algorithm for k-means clustering of time series. *Proc. Workshop on Clustering High Dimensionality Data and its Applications*. 33

[von Luxburg, 2007] von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*. 31

[Von Luxburg et al., 2008] Von Luxburg, U., Belkin, M., and Bousquet, O. (2008). Consistency of spectral clustering. *The Annals of Statistics*, pages 555–586. 31

[Wan et al., 2009] Wan, L., Ng, W. K., Dang, X. H., Yu, P. S., and Zhang, K. (2009). Density-based clustering of data streams at multiple resolutions. *ACM Transactions on Knowledge Discovery from Data*, 3(3). 37

[Wang et al., 2002] Wang, L., Mehrabi, M., and Kannatey-Asibu, E. (2002). Hidden markov model-based tool wear monitoring in turning. *Journal of Manufacturing Science and Engineering-transactions of The Asme - J MANUF SCI ENG*, 124. 33

[Wang et al., 1997] Wang, W., Yang, J., Muntz, R., et al. (1997). Sting : A statistical information grid approach to spatial data mining. In *Vldb*, volume 97, pages 186–195. Citeseer. 31

[Warren Liao, 2005] Warren Liao, T. (2005). Clustering of time series data—a survey. *Pattern Recognition*, 38(11) :1857–1874. 32, 33

[Warren Liao, 2007] Warren Liao, T. (2007). A clustering procedure for exploratory mining of vector time series. *Pattern Recognition*, 40(9) :2550–2562. 33

[Xiong and Yeung, 2002] Xiong, Y. and Yeung, D.-Y. (2002). Mixtures of arma models for model-based time series clustering. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 717 – 720. 33

[Xu and Tian, 2015] Xu, D. and Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2 :165–193. 25

[Xu and Wunsch, 2005] Xu, R. and Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3) :645–678. 25

[Yadav et al., 2019] Yadav, A. K., Garg, M. L., and Ritika (2019). Docker containers versus virtual machine-based virtualization. *Emerging Technologies in Data Mining and Information Security*, 814. 12

[Yadav and Sharma, 2012] Yadav, R. and Sharma, A. (2012). Advanced methods to improve performance of k-means algorithm : A review. *Global Journal of Computer Science and Technology*, 12(9) :47–52. 27

[Yang et al., 2009] Yang, J., Yu, K., Gong, Y., and Huang, T. (2009). Linear spatial pyramid matching using sparse coding for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1794–1801. 15

[Yogita and Toshniwal, 2013] Yogita, Y. and Toshniwal, D. (2013). Clustering techniques for streaming data-a survey. *3rd IEEE international advance computing conference*, pages 951–956. 36

[Young et al., 2019] Young, E. G., Zhu, P., Caraza-Harter, T., Arpaci-Dusseau, A. C., and Arpaci-Dusseau, R. H. (2019). The true cost of containing : A {gVisor} case study. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*. 13

[Zhang et al., 2007] Zhang, J., Chung, H. S.-H., and Lo, W.-L. (2007). Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 11(3) :326–335. 16

[Zhang et al., 2011] Zhang, J., Zhan, Z.-h., Lin, Y., Chen, N., Gong, Y.-j., Zhong, J.-h., Chung, H. S., Li, Y., and Shi, Y.-h. (2011). Evolutionary computation meets machine learning : A survey. *IEEE Computational Intelligence Magazine*, 6(4) :68–75. 16

[Zhang et al., 2010] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing : state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1) :7–18. 10

[Zhang et al., 1996] Zhang, T., Ramakrishnan, R., and Livny, M. (1996). Birch : an efficient data clustering method for very large databases. *ACM sigmod record*, 25(2) :103–114. 37