Composition sure d'API fondée sur des contrats

Anis Ahmed Nacer

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Composition sure d'API fondée sur des contrats

THÈSE

Soutenue le 09 Novembre 2021

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Anis Ahmed Nacer

Composition du jury

Président :	Isabelle Chrisment	
Rapporteurs :	Walid Gaaloul	Professeur des universités, École d'ingénieurs Télécom SudParis
	Parisa Ghodous	Professeur des universités, Université Claude Bernard Lyon 1
Examinateurs :	Philippe Lalanda	Professeur des universités, Université Grenoble-Alpes (UGA)
	Isabelle Chrisment	Professeur des universités, Université de Lorraine
Encadrants :	François Charoy	Professeur des universités, Université de Lorraine
	Olivier Perrin	Professeur des universités, Université de Lorraine

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

Mis en page avec la classe thesul.

Résumé

La virtualisation et la croissance exponentielle des offres Cloud ont transformé les capacités de calcul et de stockage informatique en commodités que l'on peut acquérir à la demande. Cette tendance a également atteint les services traditionnellement opérés pour permettre le déploiement et la mise en service d'applications de toutes sortes. Dans le cadre de ce travail, nous considérons les services middleware qui constituent les éléments de base d'une architecture logicielle. Evidemment, l'utilisation des services middleware implique de faire des choix qui doivent être guidés par des critères de qualité de déploiement des services et de coûts. À cet égard, nous sommes confrontés à deux défis principaux, à savoir la sélection des services cloud les mieux adaptés aux besoins spécifiques des architectes, et la composition qui consiste à évaluer et à sélectionner un ensemble de services atomiques répondant aux besoins applicatifs des architectes. Dans les deux cas, cette sélection doit assurer d'une part les besoins opérationnels de l'entreprise, tels que la capacité de stockage et la localisation des services, et d'autre part la qualité du déploiement des services ou des applications, comme la minimisation des coûts de déploiement tout en maintenant des performances élevées et une sécurité garantie. Cette thèse de doctorat contribue à la sélection et la composition des services cloud. En premier lieu, cette thèse propose une méthodologie pour identifier les éléments clés de comparaison et leurs relations pour la sélection des services cloud. Cette méthodologie est basée sur des données réelles disponibles auprès des fournisseurs de services, des benchmarks et des enquêtes menées auprès d'architectes en cloud. En deuxième lieu, cette thèse propose une méthodologie pour la sélection de service qui repose sur : (1) la résolution des difficultés liées à l'incapacité de déterminer comment les paramètres de configuration d'un service, appelés également Non-Functional Attributes (NFA), peuvent avoir un impact sur les exigences Non-fonctionnelles (NFR) telles que la performance, la disponibilité et la sécurité ; (2) la réalisation d'une enquête pour attribuer des poids significatifs aux NFA et aux NFR en fonction des besoins des architectes en matière d'applications ; (3) la normalisation des valeurs des NFA en fonction des types de données de ces derniers à l'aide de méthodes de normalisation adéquates ; (4) l'utilisation de la méthode d'aide à la décision Ordered Weighted Averaging Aggregation operator (OWA) pour la classification des services cloud en fonction des besoins des architectes et (5) la proposition d'un algorithme de sélection de services qui utilise les valeurs NFA normalisées et la méthode de prise décision OWA. En dernier lieu, cette thèse fournit une méthodologie pour guider les architectes dans la sélection des compositions de plans de services pour leurs architectures. Cette méthodologie repose sur : (1) les résultats de l'étude de sélection des services et sur l'analyse du processus de composition des services du point de vue des architectes ; (2) un algorithme de composition de services qui permet de générer des solutions de compositions dans un délai raisonnable, tout en adaptant la sélection des plans de service en fonction des besoins des architectes, notamment en termes de coûts. Nous avons validé l'efficacité de notre méthodologie de sélection de services

cloud en utilisant des configurations réelles de plans de services et les commentaires d'experts du domaine. Cette validation a été effectuée pour deux types de services : les services cloud de bases de données relationnelles et les services cloud de mise en file d'attente. Pour la composition de services cloud, nous l'avons validée et appliquée à la sélection de services cloud composés de deux types de services : services cloud de base de données relationnelles et services cloud de mise en file d'attente.

Abstract

Virtualisation and the exponential growth of cloud offerings have turned computing and storage capacity into commodities that can be purchased on demand. This trend has also reached the services traditionally operated to enable the deployment and commissioning of applications of all kinds. In this work, we consider middleware services that are the basic elements of a software architecture. Obviously, the use of middleware services implies making choices that must be guided by service deployment quality and cost criteria. In this respect, we are faced with two main challenges, namely the selection of the cloud services that best suit the specific needs of the architects, and the composition that consists in evaluating and selecting a set of atomic services that meet the application needs. In both cases, this selection must ensure both the operational needs of the enterprise, such as storage capacity and location of services, and the quality of the service or application deployment, such as minimising deployment costs while maintaining high performance and ensured security. This thesis work contributes to the selection and composition of cloud services. Firstly, this thesis proposes a methodology to identify key comparison elements and their relationships for the selection of cloud services. This methodology is based on real data available from service providers, benchmarks and surveys achieved with cloud architects. Secondly, this thesis proposes a methodology for service selection based on (1) resolving the difficulties associated with the inability to determine how the service's configuration parameters, also known as Non-Functional Attributes (NFA), can impact the Non-Functional Requirements (NFR) such as performance, availability and security ; (2) conducting a survey to assign meaningful weights to NFA and NFR based on the architects' needs ; (3) normalising the NFA values according to their data types using appropriate normalisation methods ; (4) using the Ordered Weighted Averaging Aggregation operator (OWA), which is a decision support method to classify cloud services according to architects' needs ; and (5) proposing a service selection algorithm that uses the normalised NFA values and the OWA decision making method. Thirdly, this thesis provides a methodology to guide architects in the selection of services plans compositions for their architectures. This methodology is based on : (1) the results of the services selection study and the analysis of the services composition process from the architects' point of view ; (2) An algorithm for services composition that generates composition solutions in a reasonable time, while adapting the selection of service plans according to the architects' needs, especially in terms of cost. We validated the effectiveness of our cloud service selection methodology using actual services plans configurations and feedback from domain experts. This validation was performed for two types of services : relational database cloud services and queuing cloud services. For cloud services composition, we validated and applied it to the selection of cloud services composed of two types of services : relational database cloud services and queuing cloud services.

Remerciements

Je tiens à remercier du fond du cœur toutes les personnes qui m'ont soutenues et aidées pour la réalisation de cette thèse.

En premier lieu, je tiens à remercier mes encadrants les professeurs François Charoy et Olivier Perrin pour m'avoir encadré et accompagné tout au long de ce projet de thèse. Merci pour tout le temps que vous m'avez accordé. Je tiens également à remercier les membres du Groupe Open pour leur disponibilité et leurs conseils avisés.

Je souhaiterai exprimer ma sincère gratitude envers tous les membres du jury pour avoir accepté d'évaluer ce travail de thèse.

Je remercie aussi l'ensemble du personnel du laboratoire pour leur gentillesse et leur bonne humeur.

Je tiens également à remercier les collègues du laboratoire LORIA-INRIA GrandEST pour les agréables moments que j'ai passé en leur compagnie. Un vif merci particulier à tous les membres de l'équipe COAST que j'ai eu la chance de connaitre pendant mes séjours au LORIA. Merci pour tous ces déjeuners et cafés animés.

Un hommage particulier à Chahrazed Labba qui m'a toujours soutenu et guidé tout au long de la réalisation de cette thèse.

Enfin, j'adresse ma gratitude et remerciement à mes parents, mes frères ainsi qu'à ma sœur. Rien n'aurait été possible sans eux.

Je dédie cette thèse à mes parents.

Sommaire

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte de travail

Le cloud computing est apparu comme un paradigme de l'informatique distribuée qui fournit des services et des ressources à la demande selon le modèle du paiement à l'utilisation [1]. Dans un tel paradigme, les fournisseurs exposent leurs offres en tant que services, qui peuvent être invoqués et composés pour implémenter des applications complexes. De nos jours, la composition de services [2] [3] [4], les API et les architectures de type microservice [5] [6] modifient la façon dont nous construisons nos applications. Ces applications se composent de nombreux composants, susceptibles d'évoluer indépendamment. Cela rend difficile l'évaluation de la qualité du logiciel résultant [7] [8]. De plus, ces applications s'appuient sur une large gamme de services middleware tels que des serveurs de bases de données, des services de messagerie, de journalisation, des services d'indexation, etc. Ces services nécessitent des personnes pour les exploiter, les configurer, les surveiller, les sécuriser et les maintenir. Cependant, à l'échelle des applications modernes, ce n'est pas rentable. Pour réduire ce goulot d'étranglement dans l'exploitation des logiciels et au-delà des systèmes d'automatisation des opérations (Puppet, Ansible), on assiste à la création d'un nombre impressionnant de services que les ingénieurs logiciels peuvent consommer pour développer et déployer leurs applications. Ils vont des services génériques comme la base de données en tant que service à des services plus spécifiques comme l'apprentissage automatique en tant que service. Les développeurs peuvent scripter ces services et les consommer tout en laissant le fardeau opérationnel à leurs fournisseurs. Ces fournisseurs de services ont la charge du maintien, de la sécurité et de la disponibilité de ces services. Par

conséquent, pour réduire les coûts de production et améliorer l'ensemble de leurs activités, les entreprises informatiques utilisent, aujourd'hui des services d'intergiciels en cloud. Ces services permettent de développer des applications plus rapidement et plus facilement. Toutefois, le principal défi à cet égard est la sélection d'un ensemble de services appropriés en fonction d'un ensemble d'attributs non fonctionnels (NFA).

L'émergence de PaaS (Platform as a Service) et SaaS (Software as a Service) a fait que les principaux fournisseurs de cloud ont maintenu leurs stratégies et leurs engagements, mais les approches de marché cloud sont toutes en mouvement. Actuellement, les fournisseurs de services cloud présentent leurs services sous forme de plans Add-ons décrits selon un ensemble d'Attributs Non Fonctionnels (NFA). Les NFA sont des paramètres qui décrivent comment un service est configuré. Ils influencent les exigences non fonctionnelles (NFR) telles que la performance, la sécurité ou la disponibilité. L'absence de standards a conduit les prestataires à décrire leurs services de différentes manières. Les descriptions des plans sont différentes, incomplètes et utilisent des désignations différentes pour le même NFA. En outre, ces descriptions ne précisent pas la relation ou l'impact entre les éléments de comparaison (NFA et NFR).

Cet environnement très concurrentiel dans le domaine du service numérique et la tendance forte à l'externalisation des aspects opérationnels font qu'il est nécessaire de s'adapter rapidement à ces mutations mais surtout au fait qu'elles ne font que commencer. Les métiers évoluent vite ainsi que les besoins des clients. Aujourd'hui un architecte logiciel doit prendre en charge toute la chaine du développement à sa dimension opérationnelle. Hors, ils ne sont pas toujours les mieux armés pour cela. C'est ce qui nous a amené à nous poser la question de l'industrialisation de cette chaine et d'anticiper l'évolution vers une externalisation de la mise en production de l'ensemble des services middleware pour supporter les applications.

Notre objectif est d'aider les architectes à trouver et à sélectionner les services au bon niveau de capacité et de qualité, ainsi que d'évaluer le coût d'exploitation de l'application avec différents scénarios de capacité et d'utilisation avec différents choix de services. Pour atteindre cet objectif, nous devons mieux comprendre le lien entre les plans des services et les exigences non fonctionnelles au niveau des fournisseurs. Nous devons également fournir un cadre permettant de comparer des services fonctionnellement similaires et un ensemble de services atomiques offrant la même fonctionnalité, sur la base des informations fournies

par les fournisseurs de services et des benchmarks disponibles.

Ce travail de recherche comporte deux dimensions. La première consiste à comprendre comment les architectes raisonnent pour exprimer leurs besoins en terme infrastructure et comment ces besoins peuvent se traduire en terme attributs des services qui vont être déployés. Cette phase est délicate parce que cette réflexion n'est pas toujours menée par les architectes mais plutôt par les ingénieurs du coté opérationnel. Dans ce travail, des interviews ont été menés pour comprendre que les choses s'expriment plus en terme priorité que besoin. La deuxième consiste à définir un modèle qui permet de faire le lien entre les qualités du déploiement logiciel et les attributs techniques des compositions de services à mettre en œuvre. Ce modèle permet d'associer des scores aux différents attributs disponibles en relation avec chaque qualité requise puis d'opérer une sélection de service et une sélection de composition de service en fonction des priorités des architectes. Ce modèle doit être validé service par service, puis pour des configurations de services complètes correspondant à des architectures de micro-services complexes. Les deux difficultés majeures concernent le calibrage des paramètres du modèle en fonction des priorités et des besoins de l'application et surtout celle due à la difficulté à obtenir des données précises de la part des fournisseurs de services. Les données disponibles sont des données utiles pour le fournisseur plus que pour le client.

Exemple de motivation

Afin de bien comprendre la problématique de recherche ainsi que l'objectif de notre travail, nous allons présenter un exemple de motivation. Supposons que nous avons une application composée de 3 micro-services. Chaque micro-service repose sur un système de base de données distinct. Une des bases de données est une instance de MongoDB et une autre est un service d'indexation. Elle comprend également un service de messagerie, un service d'authentification unique et une passerelle. Le déploiement, la maintenance et la mise à l'échelle d'une telle application peuvent rapidement devenir compliqués.

Pour la sélection des services ci-dessus, les architectes peuvent exprimer des besoins non fonctionnels (NFR). Par exemple, le service 1 doit être hautement disponible avec une performance moyenne. Ce service conserve des informations privées. Il doit être très sécurisé et la localisation des données doit être contrôlée. Le service 2 est un service commercial qui nécessite une performance élevée. Il doit également être hautement disponible. C'est

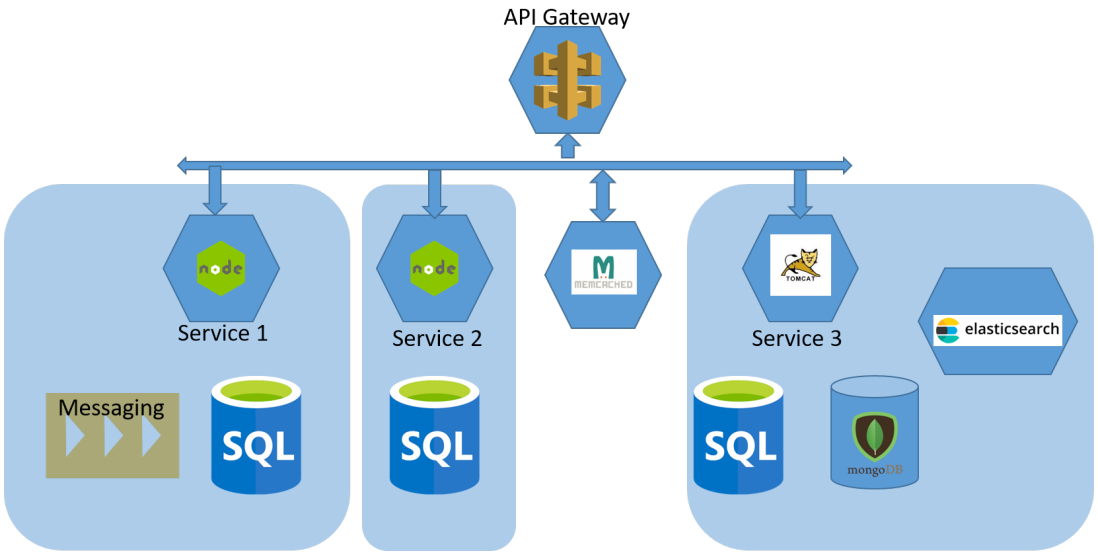

FIGURE 1.1 – Exemple d'architecture technique d'une application de micro-services

la partie centrale de l'application. Le service 3 est un service de recherche et d'indexation. Le système reste utilisable lorsqu'il n'est pas disponible. Le service de cache est disponible pour l'application globale.

Notre objectif est de permettre le déploiement de cette architecture en sélectionnant des services de fournisseurs tiers sans déployer de serveur ou de service, sauf pour les fonctions commerciales. Cela est difficile en raison de la diversité et de l'hétérogénéité du marché en termes de capacités et de prix. Les architectes sont très peu aidés pour mener leurs analyses et s'appuient surtout sur leur propre expérience. Notre objectif est d'améliorer la situation en leur fournissant un cadre décisionnel qui puisse être adapté à leurs besoins.

1.2 Problématique de recherche

La sélection des services de cloud consiste à évaluer et à choisir les services les plus adaptés aux besoins spécifiques des architectes auprès des fournisseurs de services de cloud. Quant à la composition des services, elle consiste à évaluer et à sélectionner un ensemble de services atomiques qui répondent aux besoins applicatifs des architectes auprès des fournisseurs de services cloud. Dans les deux cas, cette sélection doit garantir, d'une part, les besoins opérationnels de l'entreprise, tels que le besoin de capacité de stockage, la localisation des services, etc. et, d'autre part, la qualité du déploiement des services ou des applications, par exemple la minimisation des coûts de déploiement avec des performances élevées et une sécurité garantie.

La sélection des services cloud [9–11] et la sélection de la composition des services cloud [12–15] sont des domaines de recherche actifs qui intéressent de nombreux chercheurs. Dans le cadre de cette thèse, nous cherchons à proposer une approche efficace et optimale pour la sélection des services de cloud computing et de la composition des services de cloud computing. Comme indiqué ci-dessous, la problématique de la recherche engendre trois sous-problèmes.

1.2.1 Quels sont les élements clés permettant de comparer des services pour permettre leur sélection ?

Comme expliqué en introduction, la comparaison des services intergiciels est compliquée parce que leurs descriptions n'est pas homogène. Nous sommes face à un problème de comparaison d'éléments en présence d'informations hétérogènes et incomplètes. Dans cette section, nous allons présenter la façon dont les services sont publiés sur certains catalogues et les informations dont on peut disposer pour les comparer. Nous décrirons ensuite les critères qui peuvent servir à la sélection du point de vue des architectes.

Les fournisseurs de services dans le cloud proposent des services middleware entièrement gérés, comme des services de base de données ou des services de messagerie. Cela facilite leur utilisation pour les développeurs. Par exemple, le catalogue addon Heroku [1] propose une vingtaine de services de bases de données relationnelles managées. Chacun est disponible avec de 4 à 15 plans différents avec des frais mensuels, des capacités et caractéristiques différentes. La Figure 1.2 présente les différents plans pour le service de base de données JawsDB. Pour chaque plan, il y a une variation de capacité et d'autres attributs. Le prix est calculé sur une base mensuelle. Bien que ces plans offrent les mêmes fonctionnalités, ils diffèrent dans la manière dont ils sont déployés et gérés. Le tableau 1.1 illustre un exemple de deux services de base de données relationnelles disponibles sur la plateforme Heroku. Pour un besoin de capacité exprimé par une quantité de stockage de 40 GB, les plans sont décrits avec différents attributs non fonctionnels et une tarification différente. Il est très difficile de choisir l'un d'entre eux sur la base des descriptions disponibles. En outre, ces descriptions peuvent différer d'un plan de service à un autre. Par exemple, le service ClearDB MySQL ne présente aucune information sur le quota de RAM, le temps de rétention des sauvegardes ou le cryptage au repos.

1. https://elements.heroku.com/addons

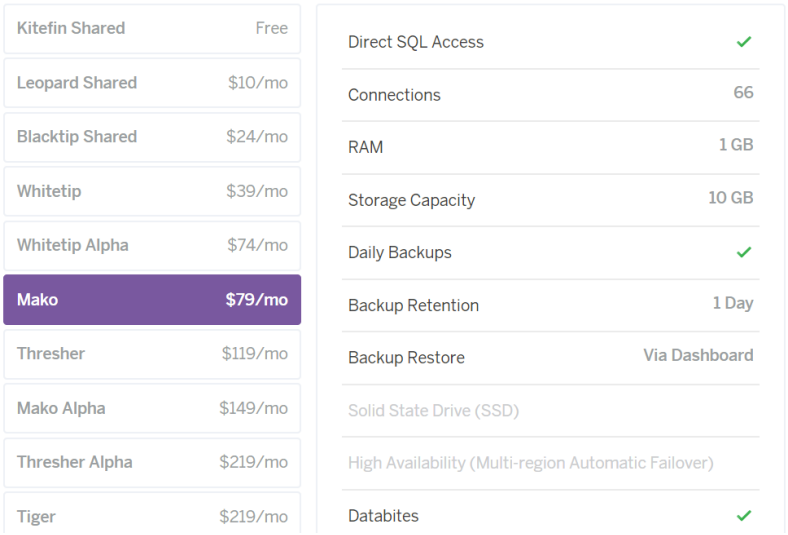

FIGURE 1.2 – Les plans de Heroku pour JawdDB

Service	attributes	Price/mo
JawsDB MySQL	Thresher Alpha : (2 GB RAM, Single Tenant, High Availability, SSD, Backup Retention 2days, Encryption at Rest)	$ 219
ClearDB MySQL	Standard 50 : (Standard US Business Day Support, Nightly Backups, Dedicated Infrastructure, Stored Procedures, Triggers and UDF's)	$ 108

TABLE 1.1 – Différents plans Elasticsearch avec différents fournisseurs

Les descriptions des services sont incomplètes. Les services ClearDB MySQL et JawsDB MySQL ne mentionnent pas le nombre d'entrées/sorties par seconde (IOPS) ni le nombre de nœuds sur lesquels le service est déployé. En outre, ces descriptions peuvent présenter des attributs sous différentes désignations. L'attribut "Dedicated Infrastructure" et "Single Tenant" ont la même signification. Il est donc difficile pour les architectes de sélectionner un plan de service particulier auprès d'un fournisseur de services cloud.

Les NFR ont un grand impact sur le fonctionnement des services. Une exigence fonctionnelle qui n'est pas spécifiée au moment de la conception d'un projet peut être ajoutée ultérieurement. Cependant, une NFR qui n'est pas correctement considérée au moment de la conception peut entraîner une modification très coûteuse ou l'annulation du projet. En outre l'importance de certains qualités attendues peut se décliner différemment selon la nature du service considéré. C'est particulièrement vrai pour des architectures à base de microservice où les contraintes sur chaque service peuvent varier. L'impact des NFR sur le projet dépend de plusieurs paramètres parmi lesquels nous citons les suivants.

1. Quel est la nature du service ? on dénombre des dizaines de catégories de services

qui peuvent être déployées pour faire fonctionner une application, de la gestion des données au monitoring en passant par les services de gestion de cache

2. Dans quel but le service est-il utilisé ? Selon l'utilisation du service, quels types d'exigences sont plus importants que les autres. Pour un service d'achat en ligne, la sécurité peut être plus importante que la performance pour la partie paiement, tandis que pour un service de jeu, la performance est plus importante que la sécurité.

Ainsi, la question de la comparaison des services cloud doit englober non seulement la spécification des exigences non fonctionnelles (NFR) et des attributs non fonctionnels (NFA) (en répondant aux questions ci-dessus), mais également la spécification de la manière dont les attributs non fonctionnels affectent les exigences non fonctionnelles. Par exemple, le nombre de connexions à une base de données a une influence positive sur la disponibilité car il détermine le nombre d'utilisateurs qui peuvent se connecter. D'autre part, il a une influence négative sur les performances car lorsque le nombre d'utilisateurs d'utilisateurs dépassent un certain seuil, le serveur se met à traiter les requêtes plus lentement.

1.2.2 Comment sélectionner des services middleware pour une architecture de micro-services ?

Outre le défi de la sélection des éléments clés de comparaison et de leurs relations, la sélection des services appropriés est confrontée aux objectifs de déploiement liés aux NFR exprimés par l'entreprise. Ceux-ci peuvent être contradictoires : par exemple, l'entreprise souhaite un déploiement de services qui assure une minimisation des coûts avec une performance maximale. La sélection des services cloud nécessite la détermination d'une valeur de qualité de déploiement (QoD) pour chaque service. Il faut prendre une décision sur la base de critères multiples et en présence de données incomplètes. C'est un problème classique mais pour lequel il faut trouver une réponse acceptable dans le contexte d'une architecture logicielle.

Pour obtenir un cadre efficace de comparaison des services, nous devons être capables de répondre à ces questions fondamentales :

1. Comment surmonter les difficultés liées à l'incapacité de mesurer l'impact des NFA sur les NFR ? Par exemple, il est difficile d'estimer le taux d'influence des processeurs d'une instance sur la performance du service. En effet, l'évaluation des

performances des processeurs des instances dépend de plusieurs caractéristiques telles que le nombre de cœurs, le type de processeur (Intel Xeon Platinum, AMD, ...) et la technologie du jeu d'instructions (AVX-512 pour Intel). L'influence de ses caractéristiques sur les performances des processeurs n'est pas connue.

2. Comment spécifier le degré d'influence des différents NFA sur les NFR ? Le choix des facteurs de pondération pour l'influence des NFA sur les NFR représente un défi majeur dans la mise en œuvre du processus de sélection des services cloud. Ce choix dépendra essentiellement du type de service que les architectes souhaitent déployer et de la nature du projet qu'ils veulent concevoir.
3. Quelle méthode de normalisation est à envisager pour résoudre le problème de l'incompatibilité des valeurs des NFA ? La normalisation des données est cruciale pour toutes sortes de problèmes de prise de décision. Comme un grand nombre de NFA sont disponibles avec un très grand écart entre leurs valeurs, le problème d'avoir des valeurs normalisées négligeables peut se poser. Ces valeurs normalisées peuvent fausser le résultat de l'évaluation du service. Par conséquent, le choix d'une méthode de normalisation appropriée aux types de données constitue un défi.
4. Quelle méthode de décision est à envisager pour le classement des services ? Le choix d'une méthode d'aide à la décision adaptée à notre problématique pour la sélection des services cloud est primordiale. Tout d'abord, cette méthode doit prendre en compte les coefficients d'importance qui expriment les relations entre les NFA et les NFR. Deuxièmement, la méthode doit résoudre le problème de la compensation entre les valeurs les plus satisfaisantes et les plus insatisfaisantes. Cela permet d'éliminer les solutions qui ont des valeurs très satisfaisantes pour des NFR ou des NFA et très insatisfaisantes pour d'autres.

Dans la littérature, plusieurs travaux ont été présentés pour la sélection des services dans le cloud. Cependant, la majorité de ces travaux sont basés sur des benchmarks [16] [17] [18] ainsi que sur des méthodes de simulation et d'hypothèse [19] [8] [20]. La première est coûteuse, longue à mettre en œuvre et n'est valable que pour des scénarios spécifiques. La seconde peut fausser l'évaluation des services en raison des valeurs utilisées, qui sont considérées comme non pertinentes par rapport aux valeurs réelles des services.

La deuxième sous problématique est la question de savoir comment sélectionner les plans de services les plus appropriés pour une architecture en se basant sur un ensemble

de besoins exprimés par les architectes et sur les données réelles disponibles auprès des prestataires de services et des travaux de benchmarks.

1.2.3 Comment sélectionner les compositions des services qui répondent au mieux aux besoins applicatifs des architectes ?

Un seul service ne peut généralement pas répondre aux exigences fonctionnelles des applications des architectes. Les architectes ont besoin d'un ensemble de services qui collaborent entre eux pour répondre aux besoins complexes de leurs applications. Il faut pouvoir sélectionner un ensemble de services en tenant compte de contraintes globales comme par exemple un coût maximum d'exploitation ou un nombre d'utilisateurs simultanés de l'application.

La figure 1.3 explique le processus de recherche et de combinaison des services. Les fournisseurs de services présentent leurs services disponibles au courtier, qui les expose aux architectes. Ces derniers envoient leurs requêtes au courtier afin que celui-ci puisse sélectionner les meilleures combinaisons de services qui répondent à leurs besoins.

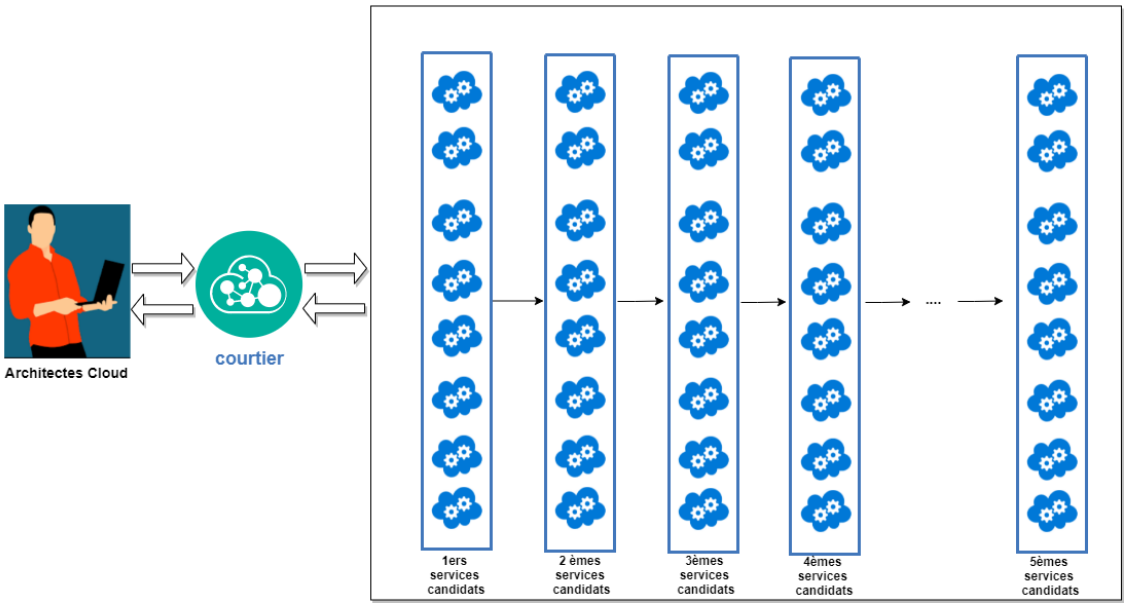

FIGURE 1.3 – Processus de composition des services cloud

L'augmentation du nombre de services offrant des fonctionnalités similaires sur le marché du cloud a entraîné une concurrence accrue entre les fournisseurs de services. Ces services similaires peuvent être situés dans des zones géographiques différentes, avec des paramètres différents en termes de qualité de déploiement des services. A cet égard, le principal défi consiste à sélectionner un ensemble de services atomiques parmi les différents

services similaires pour répondre aux exigences des architectes. En raison de la nature de l'environnement cloud, de l'évolution des caractéristiques des services et des exigences des architectes, il est nécessaire que la solution de composition des services soit conçue pour s'adapter à ces changements de manière dynamique et automatisée.

La sélection d'une combinaison de services visant à déterminer un service atomique qui réponde aux exigences fonctionnelles et non fonctionnelles des architectes est une tâche complexe. Pour y parvenir, nous devons être en mesure de relever les défis mentionnés ci-dessous.

— Comment concevoir des micro-services d'architecture pouvant servir de base pour construire des applications ?
— Quelles sont les contraintes architecturales générales ?
— Comment représenter les contraintes et les exigences techniques ?
— Quel algorithme d'aide à la décision doit être mis en œuvre ? Cet algorithme doit à partir d'un certain nombre d'exigences, de contraintes d'ordre technique, architectural et réglementaire de proposer un ensemble de services cloud composant une solution. En outre, cette méthode doit permettre de résoudre le problème de l'optimisation.

Ainsi, la troisième sous-problématique consiste à proposer une approche pour la sélection de la composition des services qui répond le mieux aux besoins des architectes.

1.3 Contributions

Nous présentons dans cette sous-section nos contributions qui abordent les limites et les sous-problèmes décrits précédemment.

1.3.1 Contribution 1 : une méthodologie pour sélectionner les éléments clés de comparaison et leurs relations

Afin de surmonter la première problématique, nous proposons une méthodologie pour identifier les éléments clés de comparaison et leurs relations pour la sélection des services cloud. Notre méthodologie est basée sur des données réelles disponibles auprès des fournisseurs de services et sur des benchmarks. Elle se déroule comme suit : **(1)** Tout d'abord, nous menons une enquête auprès des architectes afin de comprendre leurs exigences en matière

de sélection de services cloud ; **(2)** Ensuite, nous identifions les questions qui intéressent les architectes et les classons en fonction de leur influence sur les NFR ; **(3)** Puis, nous identifions les attributs des plans des fournisseurs de services qui répondent aux questions des architectes et leur influence sur les NFR à partir de la documentation technique ; **(4)** Par la suite, nous examinons les travaux sur l'évaluation des performances du cloud afin de recueillir les attributs qui influencent les performances ; **(5)** Enfin, nous menons une étude empirique pour nous assurer que la liste des attributs que nous avons identifiés était complète et pour ajouter ceux qui manquaient. Cette méthode nous donne la certitude qu'elle peut servir de base à un processus décisionnel solide qui manque totalement aujourd'hui.

Dans le cadre de nos prochaines étapes, nous utiliserons les résultats de cette étude pour évaluer les services cloud et la composition des services cloud par type d'application. Tout d'abord, l'importance relative des attributs et des NFR devra être déterminée en fonction des types d'application et l'exactitude des résultats devra être validée par les architectes. Ensuite, sur la base du résultat précédent, une méthodologie sera proposée pour évaluer les services cloud et la composition des services cloud.

Le travail présenté dans cette sous-section est un premier pas vers la sélection guidée d'un ensemble de services cloud qui répondent aux exigences d'une architecture complète.

1.3.2 Contribution 2 : une méthodologie pour la sélection de services middleware basés sur une architecture de micro-services

Nous proposons une méthodologie pour comparer les plans des fournisseurs de services de middleware afin de sélectionner les services appropriés pour une architecture d'application. Nous nous appuyons sur les résultats de l'étude présentée dans la sous-section précédente et sur les exigences des architectes en matière d'architecture d'application. Notre méthodologie repose sur : **(1)** la résolution des difficultés liées à l'incapacité de mesurer l'impact des NFA sur les performances ; **(2)** la réalisation d'une enquête pour attribuer des poids significatifs aux NFA et aux NFR en fonction des besoins des architectes en matière d'applications ; **(3)** la normalisation des valeurs des NFA en fonction des types de données de ces derniers à l'aide de méthodes de normalisation adéquates ; **(4)** l'utilisation de la méthode d'aide à la décision *Ordered weighted averaging aggregation operator (OWA)* pour la classification des services cloud en fonction des besoins des architectes. Cette méthode de décision résout le problème de la compensation, éliminant ainsi des solutions très insa-

tisfaisantes sur un NFA ou un NFR et **(5)** la proposition d'un algorithme de sélection de services qui utilise les valeurs NFA normalisées et la méthode de prise décision OWA.

Nous validons l'efficacité de notre framework de sélection des services cloud sur la base d'études de cas d'utilisation. Nous utilisons les configurations des plans des services et les avis d'experts dans le domaine pour démontrer que notre approche fournit les services adaptés aux besoins applicatifs des architectes.

1.3.3 Contribution 3 : une méthodologie pour la sélection de la composition des services en fonction du type d'application

Nous fournissons une méthodologie pour guider les architectes dans la sélection des compositions de plans de services pour leurs architectures. Cette méthodologie repose sur : (1) les résultats de l'étude de sélection des services et sur l'analyse du processus de composition de services pour une architecture technique ; (2) un processus de composition de services qui permet de générer des solutions de compositions dans un délai raisonnable, tout en adaptant la sélection des plans de service en fonction des besoins des architectes, notamment en termes de coûts.

Nous validons la pertinence de notre approche de composition de services et l'avons appliquée à la sélection de services cloud composés de deux types de services : les services cloud de bases de données relationnelles et les services cloud file de messages. Nous avons utilisé deux méthodes différentes pour cette évaluation : (i) La première consiste à évaluer chaque Template séparément. Nous utilisons différentes exigences non fonctionnelles pour les Templates et différentes exigences non fonctionnelles pour les services constituant ces Templates. Nous utilisons également deux contraintes. La première contrainte est que les services constituant les Templates doivent être basés dans une région spécifique, par exemple en Europe. La deuxième contrainte est que le coût global des services constituant le Template ne doit pas dépasser un montant prédéterminé ; (ii) La deuxième méthode de validation consiste à valider des architectures composées de plusieurs templates. Pour ce faire, nous réutilisons chacune des exigences des templates de la première méthode pour construire des architectures techniques. La contrainte à respecter est que le coût global de l'ensemble des templates constituant l'architecture ne doit pas dépasser un montant prédéterminé.

1.4 Organisation de la thèse

Le rapport de ce mémoire de thèse est organisé autour de six chapitres.

Le premier chapitre " introduction " présente le contexte de notre travail, nos problématiques de recherche et la synthèse des contributions à chacune de ces problématiques.

Le deuxième chapitre " Concepts de base " définit les concepts de base de notre domaine d'étude. Il présente, en premier lieu, le concept de cloud computing. En deuxième lieu, le concept des services dans le marché du cloud computing. Et en troisième lieu, le concept des architectures de conception des applications.

Le troisième chapitre " Etat de l'art " recouvre les différents axes de recherche qui s'apparentent à notre problématique en passant en revue les différents travaux portant sur les approches de sélection des services cloud et les approches de composition des services cloud.

Le quatrième chapitre " Identification des éléments clés des comparaisons pour la sélection des services " répond à la première problématique d'identification des éléments clés de comparaison et de leurs relations pour la sélection des services cloud à travers une méthodologie basée sur des données réelles disponibles auprès des fournisseurs de services, des analyses de benchmark et des interviews menées avec des architectes

Le cinquième chapitre " Sélection de services cloud " s'intéresse à l'évaluation des services cloud.

Le sixième chapitre " Composition des services cloud " s'intéresse à l'évaluation de la composition des services cloud.

Nous clôturons ce rapport en présentant un bilan du travail effectué et un ensemble de perspectives liées notamment à la poursuite de ce travail ainsi qu'aux nouveaux thèmes de recherche qui nous paraissent les plus pertinents.

Chapitre 2

Concepts de base

2.1 Introduction

Dans ce chapitre, nous introduisons les concepts de base liés au contexte de notre travail. Dans la section 2, nous introduisons la définition, les caractéristiques, les services, les modèles de déploiement et les qualités communes du cloud computing. Ensuite, pour une meilleure compréhension des services sur le marché du cloud computing, la section 3 présente des exemples d'IaaS, PaaS, SaaS et de solutions permettant de déployer et d'exploiter des applications de manière autonome. La section 4 présente différents types d'architecture de conception d'application cloud et leurs comparaisons. La section 5 conclut le chapitre.

2.2 Cloud computing

Dans cette section, nous définissons le concept de cloud computing et nous mettons en évidence les défis liés à son utilisation.

2.2.1 Définition

Le paradigme du cloud computing est défini de différentes manières dans la littérature, notamment dans [21] [22] [23]. Chaque définition introduit un ou plusieurs concepts selon la façon dont les auteurs considèrent le "Cloud Computing".

L'institut national des normes et de la technologie (NIST) a donné une définition qui est souvent citée comme référence. Ainsi, selon le NIST [23] :

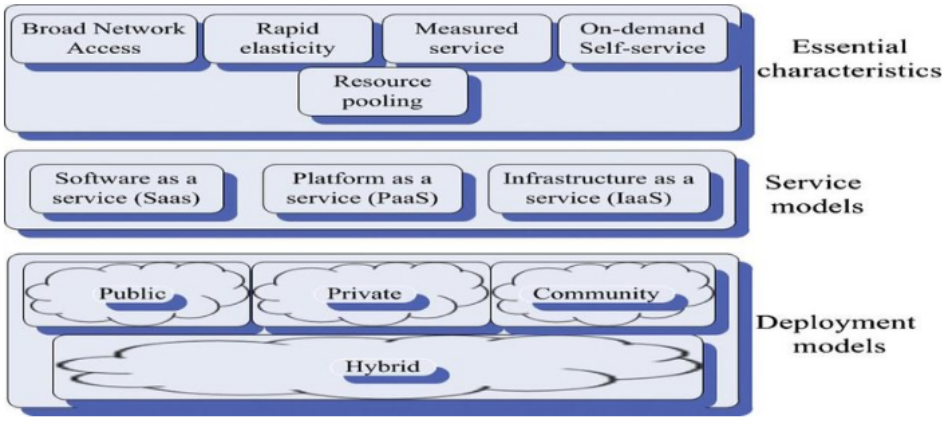

FIGURE 2.1 – NIST : définition du cloud computing

"Le Cloud Computing est un modèle informatique qui permet un accès facile et à la demande via le réseau à un ensemble partagé de ressources informatiques configurables (par exemple, réseaux, serveurs, stockage, applications, services) qui peuvent être rapidement approvisionnées et libérées avec un minimum d'effort de gestion ou d'interaction avec le fournisseur de services".

Le NIST pousse sa définition encore plus loin : Le cloud computing présente cinq caractéristiques essentielles, trois niveaux de services et quatre modèles de déploiement. Les différentes caractéristiques, services et modèles sont illustrés dans la figure 2.1.

2.2.1.1 Caractéristiques essentielles du cloud computing

— Accès aux services à la demande : le consommateur peut utiliser automatiquement les capacités informatiques, tels que la durée du serveur et le stockage. Une interaction humaine avec les fournisseurs de services n'est pas nécessaire.

— Un large accès au réseau : les services sont disponibles sur le réseau et peuvent être consultés par des mécanismes standards comme les smartphones et les PC.

— Mise en commun des ressources : les ressources sont mises en commun pour servir plusieurs consommateurs, différentes ressources physiques et virtuelles étant allouées et réaffectées de manière dynamique en fonction de la demande du consommateur.

— Redimensionnement rapide (élasticité) : les capacités sont fournies rapidement et de manière flexible, presque de manière automatique. Pour le consommateur, les capacités disponibles pour l'approvisionnement semblent illimitées et elles peuvent être achetées en toute quantité à tout moment.

— Services mesurables : les systèmes contrôlent et optimisent automatiquement l'utilisation des ressources par le suivi, le contrôle et les rapports, assurant ainsi la transparence tant pour les fournisseurs que pour les consommateurs du service utilisé.

2.2.1.2 Modèles de services cloud

Le NIST identifie trois modèles de services pour le cloud computing : Logiciels en tant que service (SaaS), plateformes en tant que service (PaaS) et infrastructures en tant que service (IaaS) [23].

— SaaS : Les fournisseurs de services de cloud proposent des applications fonctionnant sur des plateformes de cloud computing. Ces applications sont accessibles sur Internet à l'aide de différents dispositifs tels qu'un navigateur. Les SaaS sont gérés par un fournisseur de cloud computing. Par conséquent, dès qu'une erreur est détectée dans le logiciel, elle peut être facilement réparée sur l'infrastructure du fournisseur. Une telle opération est plus légère que la réparation et la mise à jour de toutes les infrastructures des utilisateurs de logiciels. Il existe plusieurs exemples de SaaS tels que Google qui propose des applications web, notamment Gmail et Google Docs, et SalesForce.com qui fournit des solutions CRM en ligne.

— PaaS : Le fournisseur de cloud computing offre une plateforme sur laquelle les utilisateurs peuvent développer et déployer leurs applications. La plateforme fournit des outils de développement, des bases de données et tout autre service pouvant aider les développeurs à construire leurs propres applications. Chaque PaaS est destiné à être utilisé dans un certain contexte, comme le développement d'applications web. Il existe plusieurs exemples de PaaS, telle que la plateforme Microsoft Windows Azure, qui peuvent être utilisés pour développer des applications (web) et des services basés sur le framework .NET et l'App Engine de Google, qui sont utilisés pour développer et déployer des applications (web) basées sur Go, Python et Java.

— IaaS : L'IaaS combine deux couches, dont la couche matérielle et la couche infrastructurelle. En effet, les fournisseurs d'IaaS fournissent à l'utilisateur une plateforme de visualisation, qui est utilisée pour installer, configurer et gérer les machines virtuelles. Dans le cadre de ce service, les fournisseurs de cloud computing sont uniquement responsables de la gestion et de la maintenance de leur matériel, tandis que

la gestion des machines virtuelles est assurée par les utilisateurs du cloud computing. Il existe plusieurs exemples d'IaaS tels que Amazon EC2 et RackSpace Open cloud.

2.2.1.3 Modèles de déploiement

Il existe quatre modèles de déploiement de cloud [23] qui sont les suivants :

— **le cloud public** est disponible pour le grand public et accessible via l'internet. Il est la propriété d'une organisation qui vend des services de cloud computing. Aujourd'hui, le marché du cloud public est en croissance continue et de nombreuses entreprises proposent ces services, comme Microsoft, Amazon et Google. Le plus grand avantage de l'utilisation de ce type de services cloud est que les fournisseurs de cloud sont responsables de la gestion de l'infrastructure. Tandis que les utilisateurs payent uniquement ce qu'ils utilisent.

— **le cloud privé** est destiné à être utilisé par une seule organisation. Ce modèle de déploiement peut être hébergé à l'intérieur ou à l'extérieur de l'organisation et géré respectivement par l'organisation utilisatrice du cloud ou par un fournisseur tiers. Le cloud privé est généralement utilisé pour traiter des données critiques et assurer des préoccupations de sécurité strictes.

— **le cloud hybride** intègre à la fois les clouds privés et publics. Les organisations utilisent un tel modèle de cloud pour traiter leurs données critiques en privé et effectuer les opérations non sensibles en public. En outre, un cloud hybride permet de surmonter le défi de l'évolutivité qui peut être induite en utilisant uniquement l'infrastructure privée et en minimisant les coûts de l'externalisation de l'ensemble des données et de l'informatique sur les ressources publiques.

— **le cloud communautaire** est disponible pour une communauté spécifique composée de plusieurs organisations, qui partagent des préoccupations communes. Cette forme de cloud peut être gérée par une ou plusieurs organisations au sein de la communauté ou peut être hébergée chez un fournisseur tiers.

2.2.1.4 Les qualités communes du Cloud Computing

Un service de cloud computing est décrit par des exigences fonctionnelles et non fonctionnelles. Les exigences fonctionnelles précisent les tâches que le service doit accomplir.

Par exemple, pour un service de stockage de bases de données relationnelles, l'exigence fonctionnelle est la capacité de stockage. Les exigences non fonctionnelles spécifient les qualités que le service doit avoir, telles que la sécurité, la disponibilité, la fiabilité, etc. Il existe de nombreux travaux dans la littérature qui décrivent les exigences non fonctionnelles. Nous présentons ci-dessous la définition des exigences non fonctionnelles qui correspondent le mieux à notre description.

— **La facilité d'utilisation** d'un service cloud est définie par les attributs de commodité d'utilisation. Les éléments telles que l'apprentissage, la capacité d'installation et la compréhensibilité peuvent être quantifiés comme étant le temps moyen que les utilisateurs du service cloud ont passé à l'utiliser, l'apprendre, l'installer et le comprendre respectivement [24].

— **La sécurité** est une exigence qui peut être appliquée au fournisseur de services de cloud ou au système développé. Dans le premier cas, le développeur de l'application est soumis aux mesures de sécurité du fournisseur, telles que les politiques d'accès à l'infrastructure physique ou les mécanismes d'isolement du réseau. Dans le second cas, la sécurité des machines virtuelles instanciées doit être assurée par l'utilisateur du cloud [25].

— **La fiabilité** consiste à fournir un service constant sans interruption ni défaillance. Elle signifie que le service doit être disponible et doit fonctionner conformément aux spécifications pour lesquelles il a été conçu.

— **La performance** représente la mesure dans laquelle un service est adapté pour supporter les charges de travail qui lui sont attribuées au fur et à mesure que celles-ci augmentent.

— **La disponibilité** est le temps pendant lequel un service de cloud est disponible. La disponibilité est proportionnelle à la fiabilité et à la maintenabilité. En effet, la maintenance d'un service de cloud est essentielle pour garantir sa disponibilité. La haute disponibilité d'un service cloud exige que les applications soient déployées avec le moins de temps d'arrêt possible. Les exigences en matière de haute disponibilité dans le cloud sont nombreuses. Il doit y avoir un équilibre entre le coût de mise en œuvre et la disponibilité. De nombreux facteurs influent sur la disponibilité : l'erreur humaine, la défaillance des logiciels, la défaillance du matériel, la migration des machines d'un serveur à l'autre et l'extensibilité des logiciels sont autant de

facteurs qui influent sur la disponibilité du service de cloud [26].

— **le passage à l'échelle** est la capacité d'un service cloud à fonctionner de manière optimale indépendamment des contraintes ajoutées en vue de répondre aux besoins des clients. Le cloud computing donne aux entreprises la possibilité d'augmenter les ressources pour répondre aux besoins de l'industrie. Les ressources peuvent être augmentées ou diminuées en fonction des besoins. Il existe différents types de passage à l'échelle, comme le passage à l'échelle verticale [27] et le passage à l'échelle horizontale [28]. Si un système est plus évolutif, un plus grand nombre d'utilisateurs peuvent en tirer profit simultanément [26].

— **Coût opérationnel** : Les services cloud peuvent être facturés de deux manières [29].

  — **Paiement à l'utilisation** : Les services de cloud ne sont payants que lorsque vous les utilisez, soit à court terme (par exemple, pour le temps processeur), soit à plus long terme (par exemple, pour le stockage dans le cloud ou le coffre-fort).

  — **Paiement sur demande** : On peut faire appel aux services cloud selon les besoins, sans avoir à faire partie de l'infrastructure informatique. C'est un avantage significatif pour l'utilisation du cloud par rapport aux services IT internes.

2.3 Écosystème du cloud computing

Les fournisseurs de services cloud visent à fournir des services aux utilisateurs par le biais du cloud computing dans un contexte opérationnel. Un fournisseur de services cloud peut être un département d'une entreprise qui fournit des services aux employés d'autres départements de celle-ci. Les fournisseurs de services peuvent fournir des services en fonction du type de déploiement du cloud. Ils peuvent fournir des services IaaS avec une capacité de calcul et de stockage de base. Ils peuvent fournir des services PaaS avec un environnement de développement d'applications, une interface de programmation, des outils de développement et une plateforme d'exploitation. Ils peuvent fournir des services SaaS assurant la maintenance et la gérance des équipements et des installations logicielles pour les utilisateurs. L'offre SaaS se présente sous la forme d'un engagement mensuel dont le coût est proportionnel à l'utilisation. Les architectes cloud sont libres de choisir le modèle de service qui répond le mieux à leurs exigences.

2.3.1 Les services cloud

Le marché du cloud computing est un marché extrêmement lucratif pour les fournisseurs de services, car de nombreuses entreprises de divers secteurs industriels adoptent de plus en plus les services cloud. Selon Gartner [30], Le marché mondial des services publics dans le cloud devrait atteindre 354,6 milliards de dollars en 2022.

Plusieurs offres diversifiées de services de cloud computing existent dans le marché pour les modèles IaaS, PaaS et SaaS. Dans cette sous-section, pour chaque modèle de cloud computing, nous expliquons ce qu'est un service en utilisant des exemples du marché du cloud computing Public.

2.3.1.1 Services IaaS

Un service IaaS publique est un service web où un utilisateur peut réserver des serveurs virtuels (également appelés "machines virtuelles") auprès d'un fournisseur de cloud computing. Ces serveurs virtuels peuvent être instanciés et gérés par le biais d'une interface web. Sur ces serveurs, les utilisateurs peuvent installer les logiciels de leur choix. Parmi ces offres IaaS figure Amazon EC2 [2] (Elastic cloud Computing). Ce service constitue la base d'autres offres qui, comme Heroku [3], utilisent son infrastructure pour fournir leurs propres services.

Une instance à la demande EC2 est une offre qui permet aux utilisateurs de réserver des serveurs virtuels et de les utiliser pour déployer leurs propres applications. Au moment de la création des instances, l'utilisateur peut sélectionner des images virtuelles préconfigurées et spécifier le centre de données où il souhaite les exécuter.

Les instances EC2 sont facturées sur une base horaire avec des tarifs différents, en fonction des caractéristiques de la machine virtuelle spécifiées par l'utilisateur (nombre de CPU et quantité de RAM), du type d'instance et du centre de données où la machine virtuelle fonctionne. La figure 2.2 montre un exemple de la configuration de ces instances.

Il convient également de noter qu'Amazon facture le trafic réseau entrant et sortant en utilisant des forfaits. De plus, Amazon facture la réservation d'adresses IP pour rendre une instance EC2 visible à l'extérieur. En outre, Amazon AWS facture les services de persistance des données. Par conséquent, pour assurer la persistance des données, un service

2. https://aws.amazon.com/fr/ec2/
3. https://elements.heroku.com/addons

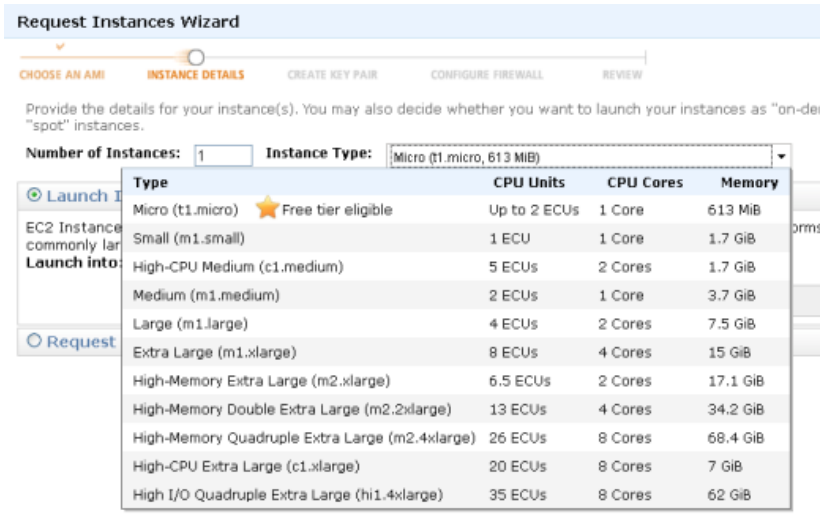

FIGURE 2.2 – Configuration des instances EC2

de bloc persistant (service S3) est nécessaire. Ainsi, pour déployer des services entièrement opérationnels, l'utilisateur devrait payer un montant forfaitaire supplémentaire pour tous ces ajouts.

Les principaux concurrents d'Amazon sont Rackspace [4] et Azure [5], qui offrent des services similaires.

Il existe d'autres types de services qui permettent de surveiller et de déployer des machines virtuelles sur l'infrastructure d'un fournisseur de cloud computing, tels que Flexiscale [6] et Rackspace. Par ailleurs, certaines solutions comme Joyent [7] offrent des services pré-paramétrés pour des services spécifiques comme SQL.

2.3.1.2 Services PaaS

Les services PaaS publics fournissent des composants cloud à certains logiciels tout en étant utilisés principalement pour des applications. Le PaaS fournit aux développeurs un cadre qu'ils peuvent utiliser pour créer des applications personnalisées. Tous les serveurs, le stockage et la mise en réseau peuvent être gérés par l'entreprise ou un fournisseur tiers, tandis que les développeurs peuvent gérer les application. Par exemple, Heroku [8] est une plateforme disponible en tant que service public et privé. Ce fournisseur de services permet aux utilisateurs de déployer des applications écrites en 9 langues sur 2 infrastructures

4. https://www.rackspace.com/cloud/servers/pricing
5. https://azure.microsoft.com/en-us/services/virtual-machines
6. https://flexiscale.com/
7. https://www.joyent.com/blog/clustrix-brings-real-time-analytics-and-scale-out-sql-to-cloud-applications
8. https://elements.heroku.com/addons

différentes. Elle dispose de 179 addons qui peuvent être intégrés dans les applications des utilisateurs. La figure 2.3 illustre un exemple d'une partie des addons Heroku de la catégorie "Data Stores".

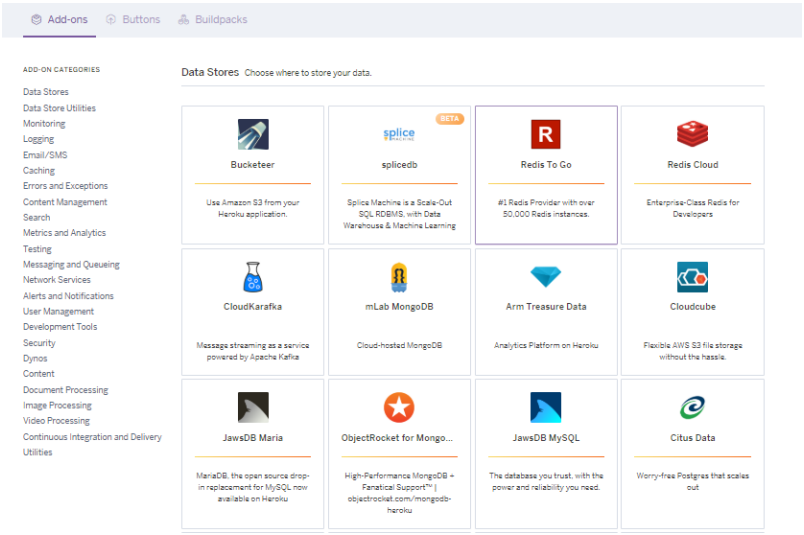

FIGURE 2.3 – Addons Heroku

Parmi les autres grands concurrents de Heroku sur le marché du cloud PaaS figurent AWS Elastic Beanstalk [9], Windows Azure [10], Force.com [11], Google App Engine [12] et OpenShift [13].

2.3.1.3 Services SaaS

Un service SaaS public est un modèle de distribution dans lequel les logiciels sont gérés et hébergés de manière centralisée par un fournisseur et mis à la disposition des clients sur la base d'un abonnement. Le SaaS est l'une des principales applications de cloud computing (autres que l'IaaS et le PaaS). Les entreprises SaaS sont présentes partout sur le marché cloud. Plus de 27 % des entreprises déclarent fonctionner entièrement en mode SaaS et 73 % envisagent de passer au SaaS complet. Les entreprises de moins de 50 salariés utilisent de 25 à 50 applications SaaS, tandis que les entreprises de plus de 250 salariés utilisent en moyenne plus de 100 outils SaaS [31].

Par exemple, amazon AWS propose plus de 150 services SaaS aux entreprises et aux particuliers pour leur fournir tous les outils dont ils ont besoin, comme les bases de don-

9. https://aws.amazon.com/fr/elasticbeanstalk/
10. https://azure.microsoft.com/en-us/free/windows-server-on-azure/
11. https://developer.salesforce.com/platform/force.com
12. https://cloud.google.com/appengine/
13. https://www.openshift.com/

nées,l'IdO (Internet des Objets), des applications professionnelles, l'apprentissage machine, le stockage, la robotique et la sécurité.

Un autre exemple est celui de Dropbox [14], l'un des principaux fournisseurs de stockage en ligne en mode SaaS qui permet aux entreprises de stocker, de partager et de collaborer plus facilement sur des fichiers et des données.

D'autres exemples populaires incluent : Google GSuite (Apps) [15], Salesforce [16], Cisco WebEx [17], SAP Concur [18] et GoToMeeting [19].

2.3.2 La composition de services cloud

La composition des services de cloud computing est le processus d'assemblage des services d'un même fournisseur ou de fournisseurs différents, sous un modèle de services de cloud computing identique ou différent, afin de construire un ensemble de services de cloud computing qui répondent aux besoins des architectes en matière d'applications.

ils existent plusieurs fournisseurs de services offrant des solution pour la composition des services en tant que produit. Parmi les solutions existantes, on peut citer Cloud Foundry [20], Docker [21], Red Hat OpenShift [22], Kubernetes [23], OpenStack [24] et Terraform [25]. L'une des solutions les plus adoptées est Cloud Foundry fournit par IBM. Cette plateforme permet aux développeurs de déployer et d'exploiter eux-mêmes des applications sans savoir comment configurer un serveur web, une base de données, un équilibreur de charge, etc. L'avantage principal de cette solution c'est qu'elle permet le déploiement d'architectures de micro-services et la mise en place d'une culture DevOps. Pour aider les architectes, elle fournit une API REST pour le déploiement, le fonctionnement et l'évolution des applications. De plus, elle peut détecter dans quel langage et dans quel cadre de programmation l'application est écrite, puis construire une application exécutable à partir des sources d'application téléchargées. Par ailleurs, elle peut mettre à l'échelle une application verticalement et horizontalement à l'aide de commandes CLI. En outre, Lorsqu'une instance

14. https://www.dropbox.com
15. https://gsuite.google.com/marketplace
16. https://www.salesforce.com
17. https://www.webex.com
18. https://www.concur.com/
19. https://www.gotomeeting.com
20. https ://www.cloudfoundry.org/
21. https ://circleci.com/docker/
22. https ://www.openshift.com/
23. https ://kubernetes.io/
24. https ://www.openstack.org/
25. https ://www.terraform.io/

d'application subit une défaillance pour une raison quelconque (par exemple, un manque de mémoire), Cloud Foundry la redémarre. Enfin, Cloud Foundry offre un marché où les systèmes tiers peuvent enregistrer leurs offres de services. Pour introduire un service sur le marché de la solution Cloud Foundry, le fournisseur de services doit fournir une API REST simple pour gérer les instances et les références du service. Dans la terminologie de Cloud Foundry, cette API est appelée API de courtier de services.

2.4 Architectures de conception des applications

Les concepteurs d'applications décident souvent de construire des architectures monolithiques ou de micro-services [32] en fonction de leurs besoins. Dans cette section, nous expliquons chacune de ces deux architectures et présentons la meilleure architecture adaptée à chaque contexte.

2.4.1 Architecture monolithique

Il s'agit d'une application à un seul niveau dans laquelle l'interface utilisateur, les règles de gestion et la couche de données sont toutes contenues dans une seule application. Cette application ne dépend d'aucun autre service ou composant. Cela est à l'opposé d'une architecture à deux ou trois niveaux qui repose sur des bibliothèques de services ou de composants qui fonctionnent indépendamment de l'application principale [33]. Par exemple, pour extraire des informations de la base de données. Les méthodes modernes utilisent un courtier d'intégration API qui s'exécute sur un autre serveur et qui généralise l'interaction avec la base de données. La monolithique consiste à établir par code une connexion directe à une couche de données, voire directement à la base de données pour traiter la requête.

Les principales faiblesses de l'architecture monolithique résident dans les points suivants :

— **Non flexibile** : Les applications monolithiques ne peuvent pas être construites en utilisant des technologies différentes.

— **Peu fiable** : La défaillance d'une partie du système entraînera la défaillance de l'ensemble du système.

— **Non évolutive** : Les applications ne sont pas facilement évolutives car elles doivent

être entièrement reconstruites à chaque mise à jour.

— **Obstacle au développement continu** : De nombreuses fonctionnalités des applications ne peuvent pas être construites et déployées en même temps.

— **Développement lent** : Le développement d'applications monolithiques prend beaucoup de temps car les fonctionnalités doivent être construites de manière séquentielle.

2.4.2 Architecture de micro-services

Le terme Microservices est généralement destiné à décrire une approche du développement de logiciels qui consiste à décomposer les fonctionnalités d'une application en composants individuels pouvant être déployés séparément les uns des autres, et qui communiquent généralement via des interfaces de programmation d'applications ou des API. L'architecture des micro-services est un type architectural qui structure une application comme un ensemble de services indépendants et autonomes. Les avantages d'une architecture de micro-services peuvent inclure :

— une amélioration de la résilience et de la tolérance aux pannes grâce à l'isolement des fonctions de service.

— une évolutivité facile avec la possibilité d'adapter les services individuellement en fonction des besoins.

— une migration plus facile puisque les services peuvent être re-architecturés ou reconstruits avec différentes technologies tout en supportant la même définition d'API.

Parmi les grandes entreprises qui ont utilisés des architectures de micro-services figurent Amazon, eBay, Gilt, Netflix, PayPal et Twitter.

Une approche de micro-services est à l'opposé d'une approche monolithique dans laquelle toutes les fonctionnalités sont intégrées dans un programme unique opérant dans un environnement unique.

2.4.3 Architecture Microservices comparée à une Architecture Monolithique

Avec l'architecture à base de micro-services, le découplage et la séparation des tâches au niveau architectural se sont généralisés. La base de code devient plus facile à gérer car chaque composant est indépendant des autres. Par conséquent, il sera plus facile d'ajouter

de nouvelles fonctionnalités aux applications. Inversement, dans une architecture monolithique, cela peut être très complexe, en particulier lorsque l'application est volumineuse. La figure 2.4 montre la différence entre une architecture monolithique et une architecture de micro-services.

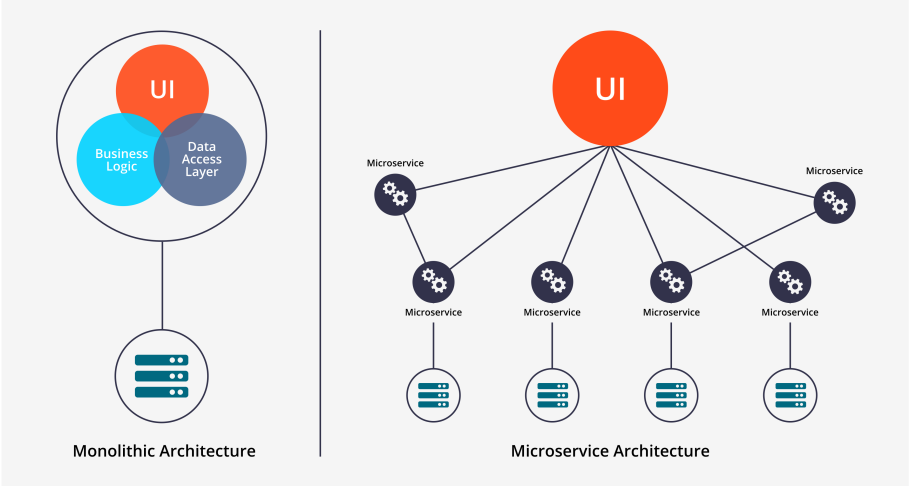

FIGURE 2.4 – Architecture Monolithique comparée à l'architecture de Micro-services.

Dans une architecture de micro-services, le déploiement de l'application est plus avantageux car les architectes construisent séparément des micro-services indépendants et les déploient sur des serveurs distincts. Les architectes peuvent donc créer et déployer les services quand ils le souhaitent sans avoir à reconstruire le reste de l'application. De plus, comme les différents services peuvent être de taille réduite et déployés séparément, il devient plus facile de les faire évoluer. Les architectes peuvent ainsi faire évoluer des services spécifiques de leur application (Pour faire évoluer une partie spécifique de l'application dans une architecture monolithique, il est nécessaire de faire évoluer l'ensemble de l'application.).

Toutefois, il est préférable de conserver l'architecture monolithique pour les applications qui ne sont pas susceptibles de devenir volumineuse. Pour les petites applications, il est simplement plus facile de rester en monolithique. En utilisant des micro-services, les architectes ont la complexité supplémentaire de distribuer des services à différents serveurs dans différents endroits, ce qui crée un effort supplémentaire car ils doivent les gérer. La mise en œuvre de micro-services aidera les architectes à long terme si l'application devient volumineuse.

2.5 Conclusion

Dans ce chapitre, nous avons présenté les concepts de base liés à notre contexte de recherche.

Dans le prochain chapitre, nous présenterons une exploration et une analyse détaillées de l'état de l'art en nous concentrant sur trois sous-problèmes de notre travail de recherche. D'abord, nous présenterons des travaux connexes axés sur les exigences non fonctionnelles dans la prise de décision architecturale. Ensuite, nous présenterons les travaux connexes axés sur la sélection des services de cloud. Enfin, nous exposerons le problème de la sélection et de l'optimisation de la composition des services cloud.

Chapitre 3

Etat de l'art

3.1 Introduction

Après avoir introduit les concepts de base liés à notre travail de recherche, nous examinons dans le présent chapitre les travaux de recherche existants dans la littérature relative à celui-ci.

Nous examinons et classons les travaux liés à notre thèse en deux catégories. Dans la section 3.2, nous présentons les travaux de recherche existants sur la sélection des services cloud. Ensuite, dans la section 3.3, nous examinons les travaux liés à la composition des services cloud. Enfin, nous concluons le chapitre dans la section 3.4.

3.2 Approches de sélection des services cloud

En raison de la concurrence entre les différents fournisseurs de services cloud, de plus en plus de services cloud ont des propriétés fonctionnelles similaires. Par exemple, Amazon [26], Google [27] et IBM [28] fournissent des services de stockage aux consommateurs, auxquels on peut accéder par le biais d'interfaces de programmation d'applications (API) basées sur des services web ou des applications spécifiques. Par conséquent, avec le grand nombre de services fonctionnellement similaires proposés par les fournisseurs, il devient plus difficile pour les utilisateurs de sélectionner le service qui répond le mieux aux besoins de leur application.

Pour identifier le meilleur service cloud pour les besoins des architectes, la qualité de

26. https ://aws.amazon.com/fr/products/storage/
27. https ://cloud.google.com/storage
28. https ://www.ibm.com/cloud/storage

service (QoS) est généralement exploitée. La QoS représente un ensemble d'attributs non fonctionnels des services tels que le temps de réponse, le débit, la fiabilité et la sécurité. Dans la pratique, l'obtention ou la mesure de la QoS n'est pas une tâche aisée. En effet, la QoS dépend fortement de l'environnement dynamique du réseau. Par exemple, différents utilisateurs de services cloud peuvent percevoir différentes valeurs de QoS pour le même service. [10]. Il est donc difficile pour les utilisateurs de services d'évaluer avec précision la QoS. Par conséquent, de nombreuses recherches ont été menées pour faciliter la sélection des services cloud afin de développer des techniques qui aideront les différents acteurs à sélectionner les services cloud appropriés pour leurs applications en fonction de la QoS.

Par ailleurs, des travaux ont porté sur la catégorisation des recherches sur la sélection des services cloud. Selon Sun et al. [34], il existe principalement quatre méthodes de classification de sélection de services : les approches basées sur la prise de décision multicritères (MCDM), les approches basées sur l'optimisation, les approches basées sur la logique et les autres approches. A la différence des travaux de la littérature qui se concentrent sur le choix de la méthode de sélection des services, notre travail s'intéresse davantage aux questions liées au traitement des critères de qualité de service et à leur pertinence pour la sélection des services cloud. Nous classons les méthodes de sélection de services cloud en trois catégories : les approches basées sur la simulation, les approches basées sur les benchmarks et les approches basées sur la confiance.

3.2.1 Sélection basée sur la simulation

De nombreux grands fournisseurs de services cloud comme Google, Oracle et Amazon vendent des services cloud qui sont gérés par des systèmes de gestion de cloud. Il est très difficile pour les particuliers et les petites institutions d'effectuer des recherches sur la sélection des services dans des environnements cloud réels en raison des coûts liés au déploiement des services. Les simulateurs cloud comme CloudSim [35] et Simgrid [36] sont donc un moyen rentable d'étudier le comportement et les performances des composants du cloud dans différentes situations et charges de travail. Dans cette section, nous abordons les travaux sur la sélection de services basée sur la simulation.

Dans [9], les auteurs ont proposé une méthode MCDM hybride floue pour la sélection et l'évaluation des services. Fuzzy-ANP [37] est utilisé pour calculer les matrices de comparaison par paire. Fuzzy-TOPSIS [38] est appliqué pour calculer les poids des critères.

La méthode Fuzzy-ELECTRE [39] est également utilisée pour classer les alternatives. Les chercheurs ont utilisé les services measurement index (SMI) [40] pour évaluer les alternatives.

Dans [11], les auteurs proposent une méthode qui utilise le processus de réseau analytique (ANP) pour la sélection de services cloud. Cette méthode repose sur le calcul de la qualité des services en fonction des préférences des utilisateurs et des attributs pondérés. Elle a été validée sur la base d'une simulation utilisant six attributs de qualité recueillis dans des recherches existantes. Cette solution, bien qu'elle constitue une innovation dans le domaine de l'aide à la décision, reste peu applicable dans un environnement réel de services cloud. En effet, elle ne permet pas de sélectionner les services en fonction de contextes spécifiques. Les auteurs ne spécifient pas d'exigences fonctionnelles pour l'évaluation des services (comme la quantité de stockage pour les services de base de données ou le nombre d'emails à envoyer pour un service de mailing). Ils simulent les exigences fonctionnelles et non fonctionnelles d'un service et évaluent ensuite ces services. Par conséquent, leur solution reste inutilisable dans un environnement réel de services cloud.

Dans l'étude [41], les auteurs ont présenté un framework basé sur les techniques combinées de la modélisation structurelle interprétative (Interpretive Structural Modeling : ISM) et du processus de réseau analytique (ANP), pour déduire l'importance de l'interaction entre une paire de critères et pour traiter les incertitudes des données. Les utilisateurs utilisent ensuite ces critères pour évaluer les services cloud.

Dans [42], les auteurs proposent un modèle de corrélation métier incluant à la fois des corrélations de qualité de service et de sélection. Ils proposent ensuite une approche pour la sélection optimale des services, basée sur la corrélation et la qualité des services, en utilisant un algorithme génétique. Ce modèle de corrélation métier est très prometteur. Cependant, il est limité à deux services abstraits tandis que nous voulons évaluer les services offerts par différents fournisseurs.

De nombreuses autres études ont été menées sur la sélection de services basée sur la simulation [43] [8] [44]. L'avantage des méthodes basées sur la simulation est de permettre de tester les services cloud avec différentes configurations et charges de travail, tout en réduisant le coût du déploiement des services. En effet, les recherches dans des environnements cloud réels sont coûteuses pour les organisations en raison des coûts associés au déploiement des services cloud. Cependant, les méthodes de simulation ont pour limite de

biaiser l'évaluation des services en raison des valeurs utilisées, qui sont considérées comme non pertinentes par rapport aux valeurs réelles des services.

3.2.2 Sélection basée sur les benchmarks

Les benchmarks des performances des services cloud est un sujet de recherche courant. Des travaux antérieurs ont déjà évalué de manière approfondie les performances de différents services cloud pour différents cas d'utilisation, et sous différentes contraintes et configurations d'expérience. Par exemple, Li et al. [17] proposent un comparateur de services cloud PaaS appelé cloudCmp , qui peut être utilisé pour comparer trois aspects spécifiques de performance (calcul élastique, stockage persistant et intra-cloud et réseau étendu) des clouds publics. Ce comparateur est intéressant car il permet l'unification du modèle de tarification et fournit des mesures de performance en fonction des besoins des utilisateurs. Dans [45], les auteurs proposent une méthodologie et un processus permettant de mettre en œuvre des benchmarks personnalisés pour tester différents fournisseurs de cloud IaaS. Grâce à cette méthodologie, toute entreprise souhaitant examiner les différentes offres de services cloud peut passer manuellement par le processus de sélection des fournisseurs, de sélection et de mise en œuvre (si nécessaire) d'une application de référence, de déploiement sur plusieurs ressources de cloud, de réalisation des tests et d'enregistrement des résultats. L'évaluation se fait à la fin des tests. Dans [46], les auteurs proposent cloudRank-D, une suite de benchmarks pour évaluer et classer les performances des systèmes de cloud hébergeant des applications big data. Dans [47], les auteurs discutent des éléments de benchmarks spécifiques aux clouds IaaS du point de vue de l'utilisateur. Ils proposent une approche générique pour les benchmarks des clouds IaaS où la gestion des ressources et des tâches est assurée par l'infrastructure de test et où les tests peuvent être effectués avec des charges de travail complexes. Leur outil SkyMark prend en charge les benchmarks dans le contexte des charges de travail multi-tâches basées sur le modèle MapReduce.

De nombreux autres travaux ont été menés dans la littérature concernant les benchmarks des services cloud IaaS [48–50]. Cependant, il est établi que les méthodes de benchmarks sont coûteuses et longues à mettre en œuvre. De plus, elles ne sont valables que pour des scénarios spécifiques et sous des contraintes spécifiques. En outre, les fournisseurs de services mettent à jour régulièrement leurs ressources logicielles et matérielles, ce qui

diminue la pertinence des résultats obtenus. Ainsi, le principal défi à cet égard réside dans le fait que les résultats ne sont pertinents que pour la période pour laquelle les benchmarks ont été effectués.

Comme la pertinence des benchmarks pour obtenir des informations sur les performances des services cloud n'est pas évidente, de nombreux autres travaux sont apparus pour analyser la stabilité des performances des fournisseurs de services cloud. L'une des premières études à grande échelle portant sur la variabilité des performances dans un environnement cloud a recueilli des mesures horaires pendant plus d'un mois et a constaté une grande variabilité d'environ 20 % pour les performances du CPU, des E/S et du réseau [51]. D'autres études ont également observé une variabilité élevée pour des instances du même type [49] [17] et ont identifié l'hétérogénéité du matériel [17, 52, 53] comme la principale cause de la variation des performances du CPU [54] au-delà du partage du CPU et du bruit dû à la multi-localisation [55, 56]. D'autres études menées au fil du temps [10, 57] ont confirmé que la variabilité des performances reste importante, en particulier pour les petits types d'instances.

Nous pensons qu'il est plus pertinent de déterminer les attributs non fonctionnels et leur influence sur les performances en se basant sur des travaux antérieurs plutôt que d'utiliser des benchmarks qui sont coûteux en particulier lorsqu'il existe un très grand nombre de services offrant des fonctionnalités similaires, valables uniquement pour des scénarios spécifiques et sous des contraintes spécifiques. Dans notre contribution à la Section 4, nous utilisons cette méthode pour déterminer les NFA qui influencent la performance des services et leurs degrés d'influence sur la performance.

3.2.3 Sélection basée sur la confiance et la réputation

Le concept de *confiance* a été introduit dans de nombreux contextes différents. Dans les environnements orientés services, Jøsang et al. [58] définissent la confiance comme *"la probabilité subjective selon laquelle un individu, A, s'attend à ce qu'un autre individu, B, effectue une action donnée dont dépend son bien-être."* La sélection basée sur la confiance dans les environnements cloud fait référence à la sélection de services basée sur l'évaluation de la fiabilité des fournisseurs ou des services cloud. Dans les approches de sélection basées sur la confiance, les modèles de confiance sont généralement proposés sur la base d'évaluations subjectives ou objectives, ou d'une combinaison de celles-ci. Les évaluations

subjectives sont généralement extraites des jugements des consommateurs ou des avis d'experts, et les évaluations objectives sont généralement obtenues à partir d'une évaluation quantitative des performances ou de descriptions de la qualité de service [59].

Il est courant d'évaluer la fiabilité d'un service sur la base des expériences et des opinions des utilisateurs de services [60] [61] [62]. Par exemple, Xu et al. [60] ont proposé un modèle de découverte de services web basé sur la réputation et la QoS, qui combine un registre UDDI amélioré pour publier les informations sur la QoS et un gestionnaire de réputation pour attribuer des scores de réputation aux services en fonction des évaluations des clients sur leurs performances. Cependant, ils n'ont pas pris en compte la crédibilité des évaluateurs. He et al [61] ont proposé une approche de gestion de la confiance, ServiceTrust, pour soutenir la sélection de services axée sur la réputation. Le ServiceTrust prend en compte la crédibilité de l'évaluateur en combinant les évaluations d'un utilisateur et d'autres évaluations personnelles pour estimer la valeur de confiance d'un fournisseur de services. Cependant, ServiceTrust modélise les interactions comme des événements binaires (succès ou échec) et ne tient pas compte de la préférence de l'utilisateur lors de la sélection des services. Noorian et al. [62] ont également évalué la valeur de confiance d'un fournisseur de services en combinant les opinions d'expérience du demandeur de services et de l'autre évaluateur, et ont proposé une méthode de sélection de services basée sur la QoS et orientée vers les préférences. Rehman et al. [63] ont proposé un framework pour surveiller et prédire dynamiquement les performances des services cloud en fonction des commentaires des utilisateurs. Cependant, il n'existe aucun mécanisme permettant de vérifier la crédibilité des utilisateurs. Dans [64], [65], Noor et al. proposent un framework pour la gestion de la confiance dans les environnements cloud et introduisent un modèle de crédibilité capable de détecter les commentaires malveillants des utilisateurs en fonction du consensus majoritaire et de la densité des commentaires. Cependant, leur travail ne prend pas en compte le cas où de nombreux utilisateurs pourraient s'entendre pour se comporter de manière malveillante. Dans [66], Ding et al. proposent un modèle appelé CSTrust pour l'évaluation des services cloud en combinant des évaluations subjectives et des évaluations objectives. Dans leur travail, certaines valeurs quantitatives de QoS manquantes peuvent être prédites en combinant les expériences d'utilisation antérieures d'autres services similaires et la satisfaction du consommateur des services cloud en ce qui concerne les attributs qualitatifs. Dans [67], les auteurs ont proposé un modèle de sélection de services cloud

basé sur des évaluations objectives et subjectives. Cependant, ce travail ne prend pas en compte les contextes d'évaluation, qui peuvent généralement affecter les évaluations. Dans la version améliorée de leurs travaux [68], les auteurs ont d'abord proposé une idée de base de calcul de la similarité de contexte dans leur sélection de services cloud, puis ils ont présenté la procédure détaillée dans [69]. Cependant, aucune de ces approches ne prend en compte la crédibilité des évaluations fournies par les utilisateurs cloud qui peuvent avoir un comportement malhonnête.

Les méthodes de sélection de services basés sur la confiance tentent d'utiliser les commentaires des utilisateurs pour évaluer la fiabilité des services [60] [61] [62]. Cependant, la plupart des méthodes d'évaluation de la confiance basées sur les notes partent du principe que chaque service (ou produit) a une valeur de réputation objective (unique) et que le but des techniques est simplement de deviner cette valeur correcte [70]. Une telle hypothèse est trompeuse dans la pratique. Par essence, la confiance est subjective et constitue intrinsèquement une opinion personnelle. Différents utilisateurs sont susceptibles d'avoir des opinions différentes sur un service spécifique en raison de leur personnalité et de leurs préférences. De plus, l'évaluation de la confiance basée sur les notes peut être soumise à des utilisateurs malveillants et à des évaluations injustifiées. Ainsi, les systèmes de confiance basés sur les notes sont également peu appropriés pour évaluer la fiabilité des services cloud. L'adaptation de cette approche pour la sélection des services cloud consiste à s'appuyer sur des architectes experts qui ont de l'expérience dans le déploiement des services cloud. Ainsi, cette approche pourrait résoudre le problème connu dans les méthodes de confiance traditionnelles en assurant l'élimination des commentaires malveillants et en obtenant des résultats réalistes puisque les architectes sont familiers avec les services cloud.

3.2.4 Synthèse

Notre revue de la littérature révèle la variété des approches de sélection de services basées sur la QoS. Ainsi, les différentes approches considérées sont les suivantes :

— Sélection basée sur la simulation [24] [9] [11] [41] [42] [43] [8] [44] : Ces approches utilisent des critères de qualité de service résultant des travaux de simulation pour évaluer les services cloud.

— Sélection basée sur les benchmarks [17] [45] [46] [47] [48] [49] [51] [49] [17] [54] [55] [56] [57] [10] [50] : Ces approches utilisent des tests de performance après le

déploiement des services pour collecter les critères de QoS permettant d'évaluer les performances de ces derniers (par exemple, le débit et le temps de réponse), ou bien elles analysent la stabilité des performances à partir des données fournies par les fournisseurs de services cloud ou de celles fournies par des travaux antérieurs pour identifier la cause principale de la variation des performances (les critères qui font varier les performances et leur influence).

— Sélection basée sur la confiance [60] [70] [61] [62] [63] [65] [66] [67] [68] [69] : Ces approches utilisent les expériences et les opinions des utilisateurs pour évaluer la fiabilité des services.

L'analyse de notre étude souligne les points suivants :

— Les méthodes basées sur la simulation permettent de tester les services cloud tout en réduisant le coût du déploiement des services. Cependant, la limite des méthodes de simulation est qu'elles biaisent l'évaluation des services. En effet, les valeurs utilisées sont considérées comme non pertinentes par rapport aux valeurs réelles des services. Pour cette raison, nous ne ferons pas appel aux méthodes basées sur la simulation pour collecter les valeurs de qualité de service.

— Les méthodes de benchmarks fournissent des résultats fiables pour évaluer la performance des services pour une configuration spécifique, pour une période de temps spécifique et sous des contraintes spécifiques. Cependant, il faut considérer que les méthodes de benchmarks sont coûteuses et longues à mettre en œuvre. Par conséquent, nous avons décidé de déterminer les attributs non fonctionnels et leur influence sur la performance en nous basant sur des travaux de benchmarks antérieurs.

— Les méthodes de sélection de services basées sur la confiance constituent un outil intéressant dans la mesure où elles fournissent un retour sur les expériences et l'opinion des utilisateurs à l'égard d'un service. Cependant, l'évaluation de la confiance basée sur les notes peut être sujette à des utilisateurs malveillants et à des évaluations injustifiées. Pour pallier à ces limites, nous proposons d'adapter cette approche en nous appuyant sur des architectes experts renommés (et non sur de simples utilisateurs) dans le déploiement des services cloud.

En guise de synthèse, nous concluons qu'il y a un manque d'approches efficaces et rentables pour sélectionner les services sur la base de critères de qualité de service pertinents.

À notre connaissance, il n'existe pas encore de méthode qui s'appuie sur les informations disponibles auprès des fournisseurs de services, les benchmarks précédents et l'expertise des architectes pour déterminer les critères de QoS et leur influence sur les NFR. Ces critères de QoS seront utilisés pour sélectionner les services adaptés aux besoins applicatifs des architectes.

Dans la prochaine section, nous présentons les travaux antérieurs sur les approches de composition de services cloud.

3.3 Approches de composition des services cloud

Une architecture est rarement basée sur un seul service. La composition des services cloud permet de combiner un ensemble de services composites en un seul service de haut niveau qui répond à des critères de qualité spécifiques des utilisateurs. La composition des services est un problème d'optimisation introduit pour la première fois par Jula et al. [3]. Selon ces mêmes auteurs, la composition des services cloud est *"le mécanisme permettant de choisir, parmi les services disponibles dans l'environnement cloud, ceux qui peuvent être combinés ensemble afin d'obtenir une composition des services dont les critères sont conformes à ceux convenus dans l'accord de niveau de service (SLA)"*.

Pour faire la distinction entre le grand nombre de services cloud similaires, la qualité de service (QoS) est considérée comme un facteur clé dans le processus de sélection et de la composition des services. Comme discuté dans la section précédente les attributs de la QoS peuvent prendre la forme de valeurs envoyées par les fournisseurs (par exemple, le temps de réponse), de commentaires des utilisateurs (par exemple, la réputation) ou de test de benchmarks (par exemple, le temps de réponse et le débit).

Plusieurs approches ont été proposées dans la littérature pour résoudre le problème de la composition des services cloud. Ces approches sont classées par [3] en 5 catégories : la première comprend les algorithmes classiques et les algorithmes basés sur les graphes [12] [13] [14] [15] [71] [72] [73] [8] [74]. La deuxième comprend les algorithmes combinatoires [75] [76] [77] [78] [79]. Dans la troisième, les algorithmes basés sur les automates sont mentionnés dans [80], la quatrième classe regroupe les structures [81], et la dernière concerne les méthodes de frameworks [82] [83] [84].

Une autre classification pour la composition des services cloud a été réalisée dans [85].

Dans ce travail, les approches de composition des services cloud sont regroupées en approches basées sur des frameworks [86] [87] [88], approches basées sur des agents [89] [90] [91], et approches basées sur des heuristiques [92] [93] [94] [95] [96] [97].

Bien que les deux études Jula et al. [3] et Vakili et al. [85] aient mis en évidence les principaux problèmes liés à la composition des services cloud, elles se limitent à classer les approches de composition des services dans un environnement mono-cloud. Le choix d'un seul cloud dans un environnement à grande échelle peut limiter les avantages des autres cloud en matière de réponse aux besoins des utilisateurs [98], étant donné que les services nécessaires pourraient être indisponibles dans un seul cloud. Par exemple, les consommateurs de cloud qui exigent un certain taux de transfert de données ont besoin de services spécifiques provenant de clouds spécifiques. En outre, certains services sélectionnés dans un seul cloud ne peuvent pas satisfaire totalement les exigences de l'utilisateur, telles que la sécurité et la conformité ou les besoins commerciaux et techniques [99]. C'est pourquoi les services utilisés dans le processus de composition doivent être combinés à partir de plusieurs clouds. Pour surmonter ce défi, une étude a été menée dans [100] pour la classification des approches permettant de composer des services à partir de plusieurs environnements cloud, tels que multicloud et intercloud. Dans cette classification, les approches proposées sont regroupées en fonction des techniques essentielles utilisées pour composer des services cloud. Quatre sous-classes sont définies dans cette étude. La première regroupe les approches basées sur des métaheuristiques, comme [101] [102] [103] [104]. La deuxième sous-classe regroupe les approches basées sur l'optimisation combinatoire, comme [98] [105]. La troisième sous-classe rassemble les approches utilisant des techniques de partitionnement de services, comme celles basées sur MapReduce dans [106] ou le clustering comme k-means dans [107] et l'analyse formelle de concepts (FCA) dans [99]. La dernière sous-classe utilise la technologie des agents comme dans [108]. Toutes les approches rassemblées dans cette étude avaient un objectif commun, qui était d'optimiser la composition des services fournis à l'utilisateur. Cela se fait en minimisant le temps d'exécution, le nombre de clouds combinés et le coût de communication inter-cloud. Chaque approche atteint l'objectif à sa manière, en fonction de la technique utilisée. Par exemple, dans les approches métaheuristiques et basées sur les agents, l'intelligence collective est adoptée pour converger, en un temps très court, vers la région optimale des clouds qui hébergent les meilleurs services. Les approches combinatoires consistent à sélectionner, dans un environnement multi-cloud, les

clouds offrant le plus grand nombre de services afin de trouver les meilleurs services dans un temps d'exécution court. Dans les approches basées sur le clustering, le regroupement des services ou des clouds les plus pertinents en fonction de leurs caractéristiques communes permet de ne garder que ceux qui sont en mesure de satisfaire les exigences des utilisateurs, c'est-à-dire de réduire l'espace de recherche à un cluster spécifique et de minimiser ensuite le temps de composition.

À la différence de la plupart des travaux de la littérature qui ne prennent pas en compte la QoS associée à la composition de services et son coût, et qui se concentrent sur le choix de la méthode de sélection de la composition des services pour minimiser le temps d'exécution [101] [98], minimiser le nombre de clouds combinés [101] [105] [102] [98] [99] et minimiser le coût de la communication inter-clouds [99], notre travail s'intéresse davantage aux questions liées au traitement des critères de qualité de service et à leur pertinence pour la sélection de la composition des services cloud. À notre connaissance, il existe très peu de travaux sur la composition des services web ou sur la composition des services cloud basés sur la QoS qui se concentrent sur l'évaluation de la qualité de la composition des services elle-même. À cette fin, nous classons les approches de sélection de composition des services en deux catégories : approche de sélection basée sur la collecte de la QoS à partir de données réelles provenant de fournisseurs de services ou de benchmarks et approche de sélection basée sur la simulation ou sur la génération de valeurs aléatoires pour le traitement de la QoS.

3.3.1 Sélection de composition basée sur la collecte de QoS à partir de données réelles provenant de fournisseurs de services ou de benchmarks

Cette catégorie de travaux regroupe les études qui ont utilisé des valeurs de qualité de service collectées sur les pages web de fournisseurs de services cloud ou par le biais de benchmarks réalisés sur des services réels. À notre connaissance, il n'existe qu'un seul travail [109] basé sur cette catégorie pour évaluer la composition des services cloud.

Dans [109] les auteurs se concentrent sur la phase de conception dans les applications basées sur le cloud IaaS, pour une meilleure composition des services. Ce problème est abordé alors que les approches existantes de composition des services cloud ne prennent généralement pas en compte, ni toutes les exigences de l'utilisateur, ni les choix de conception.

L'approche proposée combine le problème d'optimisation de la satisfaction des contraintes et les techniques de programmation linéaire mixte en nombres entiers ou de programmation linéaire par contraintes avec l'arithmétique d'intervalle. Enfin, pour l'adapter aux problèmes et variables de la vie réelle, le problème d'optimisation de la satisfaction des contraintes utilise Choco (un solveur spécifique), qui remédie au problème des contraintes non linéaires en utilisant une bibliothèque C++ appelée Ibex. Dans leur approche, les auteurs abordent plusieurs dimensions de la QoS, à savoir la performance, la sécurité et le coût. Les auteurs ont évalué les performances des fournisseurs cloud sur la base des caractéristiques et du coût des machines virtuelles (VM) en les collectant sur les pages web des fournisseurs cloud. En outre, la performance des services déployés en interne sur leurs plateformes a été déterminée par des benchmarks, ce qui a conduit à la définition des caractéristiques de leurs VMs. Par ailleurs, le coût de ces services est considéré comme nul car ils sont déployés dans une infrastructure d'hébergement préachetée. Quant au critère de QoS de sécurité, il a été évalué par rapport aux *service-level objectives (SLOs)* fournis par les fournisseurs de services sur leurs informations de sécurité. Cette solution a l'avantage d'être réaliste puisqu'elle utilise des données réelles collectées auprès des fournisseurs de services et des benchmarks des services déployés en interne. Cependant, les auteurs évaluent les performances en fonction de la demande de l'utilisateur en termes de configuration des VMs. Dans la pratique, nous ne savons pas comment les caractéristiques des VMs influencent la performance des services. Par conséquent, ce travail nécessite une étude préliminaire de la part des auteurs afin de comprendre comment les caractéristiques des machines virtuelles influencent les performances des services.

3.3.2 Sélection de composition basée sur la simulation ou génération de valeurs aléatoires pour le traitement de la QoS

Cette catégorie de travaux regroupe les études qui ont utilisé des valeurs de QoS issues de simulations ou générées de manière aléatoire pour évaluer la composition des services.

Dans [104] un Algorithme Génétique (AG) a été adopté. Dans ce travail, la composition des services web est réalisée dans un environnement distribué, afin de minimiser les violations de SLA et de combiner les services qui ont une valeur de fitness maximale. L'algorithme commence par générer une solution initiale, soit de manière aléatoire, soit en utilisant la notion de skyline. Ensuite, à chaque itération, une fonction de fitness est

utilisée pour évaluer la solution générée. Pour combiner de nouvelles populations, l'AG utilise des stratégies telles que les opérateurs de sélection, de croisement ou de mutation. Dans ce travail, les vecteurs de QoS ont été générés de manière aléatoire. Ils représentent dans l'ordre les contraintes de temps de réponse, de prix, de disponibilité et de réputation définies dans le SLA. Cette approche se caractérise par une complexité temporelle élevée.

Comme la fourniture d'une composition des services de haute qualité dans les multi-cloud n'est pas encore garantie, les auteurs ont proposé dans [107] une nouvelle méthode appelée *HireSome-II* pour garantir la confidentialité de la composition des services pour le traitement des applications de big data. Pour chaque service, les enregistrements de l'historique de la qualité de service extraits des exécutions précédentes du service sont regroupés en clusters à l'aide de l'algorithme de clustering k-means. Ce dernier est utilisé pour extraire un représentant de chaque cluster de services. Les valeurs des enregistrements de l'historique de la QoS sont générées aléatoirement pour chaque service candidat autour des cinq critères de QoS (prix d'exécution, latence d'exécution, réputation, taux de réussite d'exécution et disponibilité). Pour décrire le contexte de la composition des services, un arbre tâche-service est construit en incorporant les services candidats pour chaque tâche. La composition des services est réalisée en traitant les relations entre les tâches des utilisateurs, les services candidats et leurs historiques, tous représentés sous forme d'arbres. Quatre algorithmes sont définis par les auteurs pour décrire l'évolution, l'élagage et la régulation de l'arbre des historiques. Selon [100], la principale limite de cette approche est la consommation importante de mémoire pour le stockage de l'historique.

Pour composer des services web dans des applications cloud à grande échelle basées sur la QoS et de la qualité de l'expérience utilisateur (QoE), une méthode basée sur MapReduce est proposée par [106] comme solution pour la sélection de skyline. En raison de la complexité temporelle élevée de la sélection et de la composition des services appropriés supportés par l'utilisation des opérateurs de skyline et de MapReduce, les auteurs ont recours à l'approche de partitionnement basée sur l'élimination des blocs. Cette dernière, comme son nom l'indique, est effectivement efficace pour éliminer les blocs de services inutiles afin de raccourcir le processus de sélection. Dans leur expérience, les auteurs utilisent un ensemble de jeux de données avec des valeurs de QoS générées de manière aléatoire. Selon [100] cette approche présente toujours un problème de complexité temporelle.

Dans [110] les auteurs ont utilisé l'AG pour composer des services dans un environne-

ment multicloud géo-distribué. Pour obtenir de meilleurs résultats, l'AG utilise une version améliorée des opérateurs de croisement et de mutation, afin de composer des services en fonction des préférences des utilisateurs. Le processus de composition commence par la génération de chromosomes initiaux, qui représentent les combinaisons de services candidats, en fonction de leurs valeurs de fitness. Les chromosomes sélectionnés sont ensuite filtrés à l'aide de la roulette basée sur le rang. L'opérateur de croisement de l'AG permet de générer de nouveaux descendants à partir d'une paire de chromosomes, qui sont sélectionnés aléatoirement dans l'ensemble des candidats. Ensuite, l'opération de mutation est appliquée sur un individu choisi arbitrairement, afin de générer de meilleures solutions. La solution optimale est obtenue à l'aide d'une fonction de fitness qui évalue les chromosomes générés en fonction des niveaux de qualité de service. Dans ce travail, le contexte n'est pas précis pour évaluer la composition des services de sorte que les valeurs de QoS sont générées de manière aléatoire. Cette technique présente également une complexité temporelle élevée dans le cas d'un nombre croissant de tâches.

3.3.3 Synthése

Notre revue de la littérature révèle que la plupart des approches de composition de services existantes sont capables d'obtenir des solutions optimales [101, 102, 104, 111] et également capables de générer des services composés dans un temps d'exécution raisonnable [98, 101, 102]. Cependant, peu de travaux abordent la question de la qualité de service associée à la composition de services et son coût. De plus, les approches qui abordent cette question se basent sur la simulation et la génération aléatoire de valeurs de QoS, ce qui peut biaiser l'évaluation de la composition de services.

3.4 Conclusion

Dans ce chapitre, nous avons analysé et discuté des travaux connexes sur **(i)** les approches pour la sélection des services cloud, et **(ii)** les approches pour la sélection de la composition des services cloud. L'analyse a montré que le processus de sélection et de composition de services cloud a été largement discuté dans la littérature, mais que les approches existantes présentent certaines limites à prendre en compte pour une exploitation réelle sur le marché cloud. En effet, nous distinguons plusieurs inconvénients liés à la ges-

tion de la QoS pour la sélection et la composition de services cloud. Il s'agit notamment de l'utilisation de valeurs de qualité de service issues de la simulation ou de méthodes de génération de valeurs aléatoires. Cependant, ces méthodes peuvent fausser l'évaluation des services et de leur composition. Il s'agit également de l'utilisation des performances et des coûts uniquement pour évaluer les services cloud et leur composition. Cette évaluation des performances est effectuée à l'aide de benchmarks dont l'établissement est coûteux et prend du temps. De plus, ils ne sont valables que pour des scénarios spécifiques et sous des contraintes spécifiques. Afin de faire face à ces limitations et à celles que nous avons mentionnées en détail ci-dessus, nous nous concentrons sur une sélection guidée de services et de leur composition basée sur des critères réels collectés auprès de fournisseurs de services cloud, d'une analyse comparative pour définir l'influence des critères de QoS sur les NFR et des opinions des architectes. Nous proposons une méthode générique de sélection et de composition de services cloud qui consiste en :

— une méthode pour sélectionner les éléments clés de comparaison et leurs relations.
— une méthodologie pour la sélection de services middleware basés sur une architecture de microservices.
— une méthodologie pour la sélection de la composition des services en fonction du type d'application.

Chapitre 4

Identification des éléments clés des comparaisons pour la sélection des services

4.1 Introduction

Les fournisseurs de services cloud présentent leurs services sous forme de plans décrits selon un ensemble d'attributs non fonctionnels (NFA). Les NFA sont des attributs qui décrivent comment un service est configuré. Ils influencent les exigences non fonctionnelles (NFR) telles que la performance, la sécurité ou la disponibilité. L'absence de standards a conduit les prestataires à décrire leurs services de différentes manières. Les descriptions des plans sont différentes, incomplètes et utilisent des désignations différentes pour le même NFA. En outre, ces descriptions ne précisent pas la relation ou l'impact entre les éléments de comparaison (NFA et NFR).

Cette situation complique la comparaison des services, qui offrent des fonctionnalités similaires, en particulier avec le grand nombre de services cloud disponibles. Le choix du service le plus approprié pour répondre aux exigences non fonctionnelles des architectes devient une tâche difficile. La sélection des services cloud les plus appropriés dépend fortement du choix des éléments clés de comparaison et de leurs relations. Les éléments clés de comparaison des plans de service sont un ensemble d'exigences non fonctionnelles et d'attributs non fonctionnels des services. En pratique, une sélection incorrecte des éléments clés de comparaison et de leurs relations entraîne une évaluation incorrecte des services. Par

conséquent, les éléments clés de comparaison et leurs relations doivent être soigneusement déterminés dès le début de l'évaluation des services. Le principal défi de la sélection des services cloud est de déterminer quels sont les NFR et les NFA qui doivent être pris en compte et comment les NFA influencent les NFR. À cette fin, nous proposons de suivre la méthodologie suivante. Tout d'abord, nous devons comprendre les exigences des architectes et la façon dont ils raisonnent pour le choix des plans des services pour leurs applications. Pour y parvenir, nous proposons de mener des entretiens semi-structurés avec les architectes. Ensuite, afin de rassembler les éléments clés de comparaison qui répondent aux exigences des architectes et leurs relations, on procède comme suit. Nous passons en revue (1) les plans des fournisseurs de services, (2) les travaux antérieurs sur les benchmarks des services cloud et (3) les études qui effectuent une analyse documentaire à grande échelle pour collecter et codifier les recherches existantes sur la prévisibilité des performances des services cloud. Enfin, pour s'assurer que la liste des éléments clés de comparaison et leur relation est complète pour le processus de sélection des services, nous proposons de mener une étude empirique avec les architectes. Les résultats de notre méthodologie sont l'identification d'un ensemble NFA, NFR et leurs relations qui seront utilisés pour évaluer les services cloud.

Ce chapitre est organisé comme suit : la section 2 décrit la méthodologie proposée pour la sélection des éléments clés de comparaison et leurs relations dans le processus de sélection des services cloud. La section 3 présente la validation de notre méthodologie sur les cas d'utilisation suivants : les services cloud file de messagerie, les services cloud de bases de données relationnelles, les services cloud de gestion de cache et les services cloud d'indexation. La section 4 présente les conclusions de ce travail et les travaux futurs qui feront l'objet des prochains chapitres.

4.2 Méthode proposée

La démarche la plus importante dans le choix des services cloud consiste à identifier les exigences non fonctionnelles, les attributs non fonctionnels et leurs relations. À cette fin, nous proposons une méthode qui permet de comprendre, dans un premier temps, les préoccupations des architectes concernant le processus de sélection des services cloud grâce à des entretiens menés avec eux. Ensuite, nous tentons de répondre à leurs préoccupations en

identifiant les attributs à partir des descriptions des plans des fournisseurs de services cloud et leur influence sur les exigences non fonctionnelles. Puis, nous recueillons les attributs qui influencent les performances en examinant les travaux antérieurs sur les benchmarks. Enfin, nous menons une étude empirique avec les architectes afin de comprendre leurs exigences en matière de services et de nous assurer que la liste des NFA et des NFR collectés est complète et pertinente pour la comparaison des services. La figure 4.1 donne un aperçu de notre méthodologie.

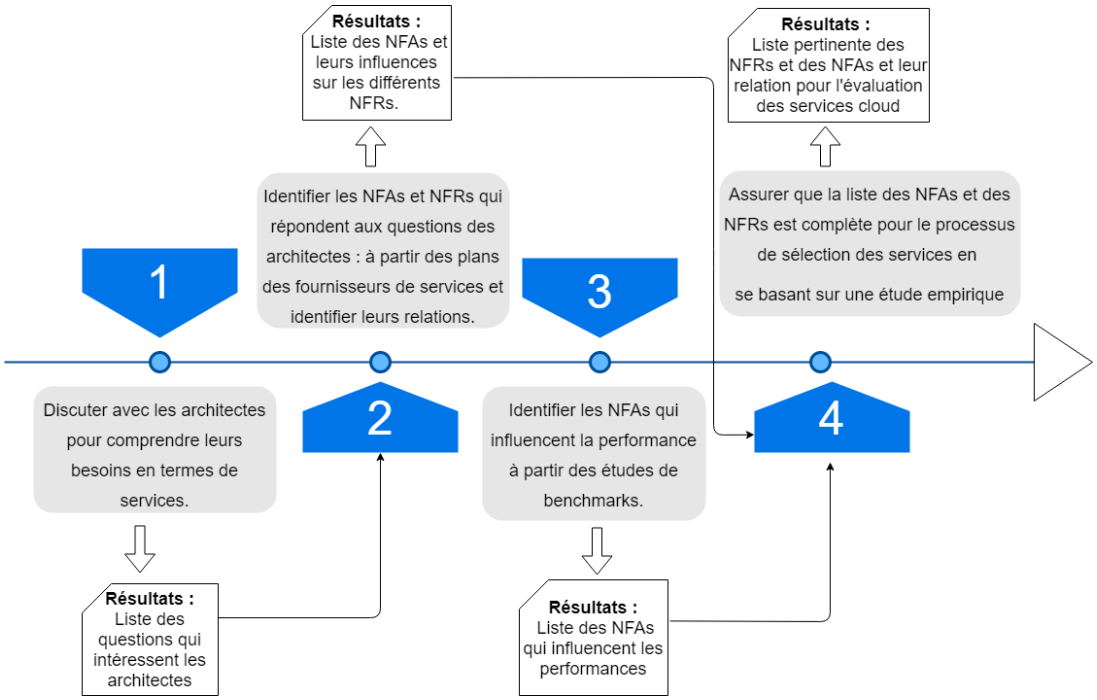

FIGURE 4.1 – Méthodologie d'identification des NFA, des NFR et de leurs relations pour le problème de sélection des services cloud

4.2.1 Identification des besoins des architectes

Afin de sélectionner les plans des services appropriés pour une application spécifique, nous devons comprendre les exigences des architectes. C'est pourquoi nous avons organisé des entretiens avec certains d'entre eux afin de clarifier leurs exigences minimales et leurs attentes vis-à-vis différents services. Nous avons interrogé les architectes sur leurs exigences globales en matière de services au moyen de questions ouvertes. Par exemple, lors du déploiement de leurs applications, quelles sont les questions qui les préoccupent en ce qui concerne le choix des plans des fournisseurs de services cloud ? À partir de ces entretiens, nous avons établi une liste de questions que les architectes prennent en compte lorsqu'ils choisissent des services cloud, notamment les suivantes à titre d'exemple : com-

ment prévoir la capacité ? Les spécifications de performance sont-elles alignées par rapport aux attentes des architectes en matière de services cloud ? Les plans de service prévoient-ils une réplication et un support de basculement ? Comment le service est-il déployé ?

Ce travail répond à un double objectif :

— il regroupe les questions communes pour l'évaluation générique des différents services offerts par les différents fournisseurs.

— il définit les NFA qui servent à évaluer les services. Les questions sont regroupées en fonction de leur influence sur ces NFR.

La prochaine sous-section analyse les plans de plusieurs fournisseurs de services cloud (Amazon, Azure, Google, Heroku, IBM) afin de recueillir les NFA qui répondent aux questions des architectes.

4.2.2 Identification des attributs des plans de service

Les fournisseurs de services donnent peu d'informations sur les conditions d'exploitation des services. Dans cette section, nous identifions les attributs qui permettent d'évaluer et de faire correspondre les plans de services de cloudware aux exigences des architectes.

Selon les plans, les fournisseurs de services indiquent la présence ou l'absence d'une certaine configuration opérationnelle. L'évaluation des plans à une échelle absolue reste difficile. Nous avons identifié les attributs associés aux plans des services des fournisseurs qui donnent accès à leurs services sur Heroku [29], Amazon AWS [30], Google Cloud Platform [31], IBM Cloud [32] et Microsoft Azure [33]. Nous les avons classés en 4 catégories : attributs de capacité, attributs fonctionnels, attributs de service et attributs techniques.

— Les attributs de capacité fournissent des informations sur la performance et la capacité du service. Nous pouvons les utiliser pour calibrer les plans en fonction des exigences de l'application. Parfois, la capacité est exprimée en termes qui ne correspondent pas directement aux exigences de l'application. Par exemple, IOPS (Input/Output per second) donne des informations de bas niveau sur la performance du disque mais pas sur la performance du système de base de données utilisant ce disque. Les attributs de capacité d'un service de base de données comprennent, par

29. https://elements.heroku.com/addons
30. https://aws.amazon.com/fr/
31. https://cloud.google.com/gcp/
32. https://www.ibm.com/cloud
33. https://azure.microsoft.com/fr-fr/

exemple, le nombre maximum de connexions simultanées à la base de données, le nombre d'IOPS et la capacité de stockage en Go.

— Les attributs fonctionnels donnent des informations sur la couverture fonctionnelle du service. Un service de base de données peut, par exemple, inclure des fonctions supplémentaires, notamment la surveillance de certaines caractéristiques (comme le débit des requêtes) ou l'audit des fonctions.

— Les attributs de service fournissent des informations sur les accords de niveau de service (SLA [34]) tels que le SLA sur la disponibilité du support (24/24 et 7/7), le SLA sur la disponibilité du service et les protocoles pris en charge par le service.

— Les attributs techniques sont tous les autres attributs qui décrivent comment le service est déployé ou exploité. Ils ont un impact différent sur les NFR du service.

Pour déterminer l'impact des NFA sur les NFR, nous avons d'abord discuté avec les architectes de leurs expériences en matière de déploiement de services. Par exemple, ils ont mentionné que l'utilisation d'une instance Multi-AZ pour leurs bases de données relationnelles a dégradé considérablement les performances du service. Par conséquent, nous considérons que la réplication Multi-AZ a un impact négatif sur les performances. De plus, nous avons examiné la documentation technique des services cloud pour comprendre les raisons pour lesquelles les fournisseurs mettent en œuvre chacun des NFA pour leurs services. De cette façon, nous comprenons leur influence sur les NFR des services. Par exemple, la réplication Multi-AZ permet un basculement automatique vers l'instance de sauvegarde lorsque l'infrastructure est défaillante [112]. Cela permet à l'application de récupérer les opérations de la base de données sans interruption et sans perte de données. Par conséquent, la réplication Multi-AZ influence positivement la disponibilité et la fiabilité.

Il reste difficile de déterminer le degré d'influence des NFA sur les NFR. Il dépend des facteurs dont les informations ne sont pas disponibles auprès des fournisseurs de services. Cela inclut le déploiement de l'architecture du fournisseur : le nombre de nœuds sur lesquels le service est déployé et les technologies utilisées pour développer le service. En outre, l'influence des NFA sur les NFR dépend de la nature des applications auxquels le service est destiné. Pour une application bancaire, la perte d'une transaction serait préjudiciable à l'entreprise, tandis que pour une application de jeu en ligne, elle serait moins importante. En raison de ces difficultés, nous n'indiquons que l'impact positif, neutre ou négatif des

34. Service Level Agreement

attributs techniques (+,=,-). Le tableau 4.1 donne un exemple des attributs techniques d'un service de base de données et de leurs impacts possibles sur les NFR.

Attributs	Performance	Disponibilité	Fiabilité	Sécurité	Scalabilité
RAM	+	=	=	=	=
IOPS	+	=	=	=	=
Backup	-	+	+	=	=
Rollback	=	+	=	=	=
Replication	-	+	+	=	=

TABLE 4.1 – Exemple de l'influence de certains NFA sur les NFR pour les services cloud de bases de données relationnelles

Dans cette section, nous avons montré comment dresser une liste de NFA à partir des plans de services cloud et leurs influences potentielles (positives ou négatives) sur les différents NFR. Par exemple, nous avons identifié les attributs des services de base de données SQL tels que la RAM, IOPS, Backup, Roolback et la réplication ainsi que leurs influences sur les différents NFR. La réplication, par exemple, a une influence négative sur les performances et a une influence positive sur la disponibilité et la fiabilité. La prochaine section passe en revue la documentation sur les benchmarks des services cloud et la revue de la littérature sur les benchmarks des services cloud afin de recueillir les attributs qui influencent la performance.

4.2.3 Identification des attributs sur la base de benchmarks

La plupart des exigences non fonctionnelles, à l'exception de la performance, peuvent être évaluées à partir des attributs non fonctionnels fournis par les fournisseurs de services cloud. La performance doit être considérée comme un NFR particulier. La seule façon d'identifier les attributs qui ont un impact sur ce NFR, est de se référer à des benchmarks. Afin de comprendre les attributs qui influencent la performance des services, nous avons examiné les travaux de benchmarks [113] qui comparent la performance de différents services cloud pour différents cas d'utilisation, sous différentes contraintes et configurations expérimentales. Cette étude est intéressante mais incomplète car la plupart des travaux de benchmarks ne fournissent pas tous les attributs qui influencent la performance dans leurs expérimentations. Dans les expérimentations, pour un même cas d'utilisation, avec les mêmes contraintes et configurations expérimentales, nous avons observé que les résultats des performances sont différents. Nous ne savons pas si cela est dû à des facteurs externes

(par exemple, la performance des fournisseurs de services cloud changeant au fil du temps), à des attributs non déclarés ou à des inexactitudes techniques. Pour compléter notre étude, nous avons passé en revue les travaux antérieurs qui se sont concentrés sur une analyse structurée des principes fondamentaux de variation et de prévisibilité des performances des fournisseurs de services. Cette étude s'est appuyée sur des analyses comparatives et des benchmarks antérieurs afin de valider les résultats des conclusions auxquelles ils peuvent aboutir. Par exemple, à partir de l'étude [10], nous avons conclu que les différents jours de la semaine, les différentes heures de la journée et la localisation géographique influencent la performance des services à des degrés variables pour les différents fournisseurs.

Dans cette sous-section, nous avons identifié un moyen de rassembler les attributs qui influencent la performance. La sous-section suivante présente une étude empirique pour s'assurer que tous les attributs nécessaires à l'évaluation des services cloud ont été collectés.

4.2.4 Sélection des NFA et des NFR en tenant compte des besoins des architectes

Afin de s'assurer que la liste des attributs techniques que nous avons collectés est complète et pertinente pour l'évaluation des services cloud, nous proposons de mener une étude empirique pour comprendre comment les architectes décident des types de composants qu'ils utilisent pour développer une nouvelle application, et quels critères guident leur choix. Nous nous sommes particulièrement intéressés aux composants de base (base de données, messagerie, cache, indexation, surveillance, etc.) qui peuvent être déployés comme serveurs ou consommés comme services dans le cloud. Nous avons mené des entretiens semi-structurés, comme ceux réalisés dans [114] [115], sur une perception générale des services avec sept architectes logiciels. Tous les architectes ont plus de quatre ans d'expérience dans les environnements de cloud computing et ont participé à plusieurs projets couvrant différents domaines d'application : ingénierie des données, développement de logiciels/systèmes, systèmes d'ingénierie, infrastructure de test, systèmes de conception architecturale à l'échelle, gestion de projet. Nous avons mené des entretiens individuels, ayant duré environ une heure pour chacun d'eux. Les entretiens sont enregistrés, transcrits et synthétisés. Les réponses des architectes sont évaluées sur la base de leurs connaissances techniques et de leur expérience. L'enquête pour cette étude est construite sur 37 questions divisées en trois parties : des questions sur l'expérience de l'architecte, des questions sur

l'application et des questions sur chaque service. Ces questions sont présentées dans l'annexe A. Par exemple, lorsqu'on demande aux architectes : "Pourquoi voulez-vous passer du service de base de données du fournisseur IBM à Amazon pour votre application ? Les architectes A1, A2, A3, A4 et A6 ont répondu : "*IBM ne fournit pas de Virtual Private Cloud (VPC) pour ses services et cela nous motive à passer aux services d'Amazon*" et les architectes A5 et A7 ont répondu : *"Nous considérons que le VPC est important, mais pas au point de faire migrer notre service vers un autre fournisseur de services cloud"*.

Une deuxième question pour les architectes est la suivante : "Comment utilisez-vous les sauvegardes de votre service de base de données dans votre application ? Les architectes ont répondu de la même manière : "la fréquence des sauvegardes doit être d'une fois par jour, la conservation des sauvegardes doit dépasser une semaine et nous avons besoin d'une sauvegarde manuelle en cas de migration". A partir de ces réponses, nous concluons que les backups et leurs configurations peuvent être des NFA importants pour l'évaluation des services cloud et ce, en fonction des besoins des applications des architectes. Nous avons posé les mêmes types de questions sur la sélection des services et avons recueilli les attributs non fonctionnels qui manquaient pour l'évaluation des services, en nous assurant en même temps que la liste des attributs non fonctionnels que nous avions recueillie était complète pour l'évaluation des services.

Dans cette sous-section, nous avons filtré les NFA qui ont été recueillis lors des étapes précédentes et nous n'avons retenu que ceux qui étaient pertinents pour l'évaluation des services. Par exemple, les SLAs ont été éliminés des NFA utilisés pour évaluer les services cloud. En effet, le respect des SLAs par les fournisseurs de services n'est pas évident. En plus, il est difficile de savoir avec certitude si les SLAs ont réellement un sens, ou s'ils ne sont qu'une technique de marketing pour attirer les clients. En outre, nous avons identifié les attributs manquants qui ne pouvaient pas être collectés lors des étapes précédentes. Par exemple, nous avons identifié que le VPC est important pour la sécurité des services de base de données.

A partir de l'utilisation de notre méthodologie, nous visons à identifier une liste pertinente des NFA, des NFR et de leurs relations pour lesquelles les services cloud seront évalués.

4.3 Validation empirique de la méthode

Pour valider l'approche décrite dans la section 4.2, nous avons réalisé des études de cas suivant cette méthode pour différents services cloud : les services cloud de base de données relationnelles, les services cloud file de messagerie, les services cloud de gestion de cache et les services cloud d'indexation. Ces services sont largement disponibles et largement utilisés dans l'architecture des applications. Ils sont utilisés en particulier dans les contextes ci-dessous.

— **Les services cloud de base de données** sont utilisés pour sauvegarder, accéder et partager des données ou d'autres tâches.
— **Les services cloud file de messagerie** sont utilisés dans les architectures de micro-services pour la communication entre services.
— **Les services cloud de cache** sont utilisés pour augmenter les performances de récupération des données au niveau des micro-services.
— **Les services cloud d'indexation** sont utilisés pour aider les entreprises à analyser et à rechercher rapidement de grands volumes de données.

Nous examinons ces services sur la base des informations de configuration réelles disponibles auprès des fournisseurs de services et des travaux de benchmarks. Nous soulignons que la localisation géographique, où le service est déployé, constitue un besoin fondamental dans toutes nos évaluations des services.

4.3.1 Étude de cas 1 : services cloud de bases de données relationnelles

L'administration des bases de données est une tâche coûteuse et qui prend beaucoup de temps. Pour réduire ce coût, les fournisseurs de services cloud tels qu'Amazon, Azure et Google proposent des services de bases de données gérées sous différents types de plans. Ces plans sont décrits par divers NFA et diffèrent d'un fournisseur à l'autre, sont incomplets ou identiques avec des désignations différentes. En outre, il est difficile de savoir comment ils influencent les NFR. Les architectes ne savent plus quelles sont les bases pour évaluer les plans de services des fournisseurs. Dans cette section, nous appliquons notre méthodologie pour identifier les éléments clés de comparaison et leurs relations pour les services cloud de bases de données relationnelles.

4.3.1.1 Identification des besoins des architectes

Dans cette section, nous présentons les questions, soulevées par l'étude introduite dans la sous-section 4.2.1.

Notez que les résultats de cette étude sont génériques et peuvent être réutilisés pour tous les types de services. Cette étude empirique a regroupé une liste de questions qui intéressent les architectes et les a classifiées en fonction de leurs influences sur les différentes exigences non fonctionnelles. Les résultats complets de cette étude sont décrits ci-dessous.

1. Capacité
   (a) Comment prévoir la capacité ? Par exemple, pour une base de données : nombre de connexion à la base de données, capacité de stockage, nombre d'index, etc.
   (b) Les architectes disposent-ils de mesures de gestion de la capacité pour ce service ? Pour quelles mesures ? Comment les architectes peuvent-ils être informés des mesures annoncées ? (seuils, recommandations, prévisions, etc.).
2. Performance
   (a) Les spécifications de performance sont-elles alignées avec les exigences des architectes pour les services cloud ?
   (b) Comment peut-on surveiller les performances ?
3. Disponibilité
   (a) Les plans de service prévoient-ils une réplication et une assistance en cas de panne ? Comment le service est-il déployé ? Est-il déployé dans plusieurs régions ou seulement dans une seule ? Dans combien de zones de disponibilité ?
   (b) Le service fournit-il un plan de reprise après sinistre ?
   (c) Est-il conforme au plan de continuité de l'application ?
   (d) Le service peut-il être pris en charge lorsqu'un centre de données est complètement perdu ?
4. Fiabilité
   (a) Quelle est la fréquence/gravité des défaillances de service ?
   (b) Quel est le temps moyen entre deux défaillances (MTBF) ?
   (c) Quel est le temps moyen de réparation du système (MTTR) ?

(d) Est-ce que mon service fournit de la sauvegarde et de la restauration ?

5. Sécurité

   (a) Est-ce que le service dispose d'une politique de sécurité et de confidentialité documentée ?

   (b) Comment recevoir des notifications en cas d'incidents et d'intrusions dans le service ?

   (c) Comment procéder à l'expertise juridique ?

   (d) Quel est le modèle de tenant proposé pour le service ?

   (e) Les données sont-elles chiffrées au repos et en transit ?

6. Scalabilité

   (a) Le service est-il extensible en mode automatique ou manuel ?

   (b) Quel type d'extension (horizontal ou vertical) est fournie par le service ?

7. Interopérabilité

   (a) Est-ce que ce service peut être combiné avec des outils et d'autres services qui intéressent les architectes ? Cette question soulève des défis de nature technique. Il est nécessaire de pouvoir communiquer le service que les architectes veulent déployer avec différentes technologies, disponibles en différents endroits, sur des architectures hétérogènes.

8. Portabilité des données

   (a) La migration est-elle prise en charge par le service ? Combien de temps dure cette opération ?

   (b) Le service dispose-t-il d'une politique de migration documentée ?

9. Contrats

   (a) Les fournisseurs sont-ils en mesure de satisfaire les exigences du SLA de l'architecte pour un service ?

   (b) Les pénalités sont-elles importantes lorsque les exigences des SLAs ne sont pas respectées ?

10. Législation et conformité

    (a) Existe-t-il des contrôles et des fonctions d'audit adéquats pour répondre aux exigences légales et les exigences de conformité pour le service ?

(b) Quelles sont les régions disponibles pour le service ? Sont-elles adéquates en termes d'exigences légales et réglementaires ?

11. Coût

(a) Comment obtenir le service le moins cher qui répond le mieux aux exigences des architectes ?

L'objectif de cette étude est de rassembler les NFA qui répondent aux questions des architectes figurant dans les plans des fournisseurs de services.

4.3.1.2 Résultat de l'identification des attributs des plans de service et de leurs influences sur les NFR

Pour valider notre méthodologie, nous détaillons les attributs des services de bases de données relationnelles. De nombreuses solutions sont disponibles pour ces services et sont déployées par différents fournisseurs de services. Les attributs intéressants des services de bases de données relationnelles sont les suivants.

Les attributs de capacité changent en fonction du service de cloud. Pour les services cloud de bases de données relationnelles, les seuls attributs de capacité qui peuvent être identifiés sont :

1. Capacité de stockage : la quantité de données pouvant être stockées en Go
2. IOPS correspondant à la performance du disque dur de la machine sur laquelle le service fonctionne.
3. Le nombre maximal de connexions simultanées à la base de données

La surveillance et l'audit sont les seuls attributs fonctionnels que nous avons pu identifier.

Les attributs techniques sont plus divers et hétérogènes et peuvent avoir un impact différent sur les attributs non fonctionnels. Comme expliqué précédemment, à l'exception de la performance, pour laquelle une analyse de benchmarks supplémentaire peut être nécessaire, il n'y a que deux façons de déterminer l'impact des NFA sur les NFR des services cloud :

— utiliser l'expertise des architectes qui ont déployé le service.
— consulter la documentation technique [116] [117] [118] sur la raison pour laquelle un NFA a été implanté par le fournisseur de services et comment il influence les NFR

du service.

Nous avons appris, par exemple, que les instances de base de données utilisant des déploiements Multi-AZ dégradent les performances parce qu'elles peuvent augmenter la latence d'écriture et de commit par rapport à un déploiement Single-AZ, en raison de la réplication synchrone des données qui se produisent.

Face à la difficulté de déterminer le niveau d'influence des NFA sur les NFR (comme expliqué dans la section 4.2.2), nous présentons dans le tableau 4.2 les attributs techniques ayant l'impact positif, neutre ou négatif estimé des NFA(+,=,-). À ce stade, c'est tout ce que nous pouvons dire sur cet impact.

Attributs	Performance	Disponibilité	Fiabilité	Sécurité	Scalabilité
RAM	+	=	=	=	=
IOPS	+	=	=	=	=
SSD	+	=	=	=	=
Server cores	+	=	=	=	=
Backup	-	+	+	=	=
Rollback	=	+	=	=	=
Replication	-	+	+	=	=
Single tenant	+	=	=	+	=
Multi tenants	-	=	-	+	=
Encryption	=	=	=	+	=
VPC	=	=	=	+	=
Multi region	-	+	+	=	=
Scaling Auto increase	=	=	=	=	+
Scaling not supported	=	=	=	=	-
Scaling converted to other storage types	=	=	=	=	+
Read replicas	-	=	=	=	+

TABLE 4.2 – Influences des NFA sur les NFR pour les services cloud de bases de données relationnelles

Dans cette sous-section, nous avons appliqué notre méthode pour répondre aux questions des architectes (soulevées dans la sous-section 4.3.1.1) en dressant une liste des NFA à partir des plans des fournisseurs de services cloud de bases de données relationnelles et en identifiant leurs influences (positives, négatives ou neutres) sur les différents NFR.

4.3.1.3 Résultat de l'identification des attributs sur la base de benchmarks

Les services cloud de bases de données relationnelles s'exécutent sur les machines virtuelles des fournisseurs. Ces machines virtuelles ont un impact direct sur les services de bases de données relationnelles qui tournent dessus. Par conséquent, les NFA qui influencent

les performances des machines virtuelles influencent également les performances des services de bases de données relationnelles. Nous avons donc passé en revue les travaux antérieurs [10] qui consistaient à mener une analyse documentaire à grande échelle pour collecter et codifier les recherches existantes sur la prévisibilité des performances des cloud publics IaaS (Infrastructure as a Service). Nous avons conclu que les attributs suivants sont pertinents pour évaluer la performance des services cloud de bases de données relationnelles (car ils affectent la performance du service) :

— le modèle de CPU, l'hétérogénéité matérielle et le modèle de tenant.
— les facteurs temporels (heure et jour de la semaine) et les facteurs géographiques (région).
— la localisation de l'utilisateur.
— le nombre de nœuds et le nombre de conteneurs sur lesquels le service est déployé.

Pour compléter notre étude, nous avons passé en revue les précédents travaux d'évaluation des performances qui ont été réalisés directement sur les services cloud de bases de données relationnelles [119] [120] [113]. Nous avons collecté les attributs suivants en plus de ceux que nous avons précédemment collectés sur les performances des services cloud de bases de données relationnelles :

— les connexions des utilisateurs sont une ressource qui peut limiter les performances.
— la réplication et le basculement réduisent considérablement les performances du service.
— l'emplacement du centre de données influence les performances.

Dans cette sous-section, nous avons collecté la liste des attributs qui influencent la performance des services de bases de données relationnelles cloud.

4.3.1.4 Résultat de la sélection des NFA et des NFR à partir d'une étude empirique

Nous avons mené une étude empirique avec les architectes sur les services cloud de base de données relationnelles, comme mentionné dans la sous-section 4.2.4, et recueilli d'autres réponses que celles mentionnées dans cette même sous-section. Par exemple, à la question "Pourquoi avez-vous choisi l'option de haute disponibilité (HA) pour votre service alors qu'il coûte environ deux fois plus cher qu'une instance ordinaire ? les architectes A1, A2, A5 et A6 ont répondu : *"Le service MySQL est un service métier dans notre application. Il*

devrait être très disponible. C'est la partie critique de l'application. Une défaillance de ce service entraînera la défaillance de toute l'application. Un basculement est nécessaire pour maintenir ce service disponible lorsque l'instance est "bloquée". Il est arrivé que l'instance principale soit "bloquée" et qu'un redémarrage de l'instance prenne jusqu'à 30 minutes (ticket fournisseur). Pour éviter ces temps d'arrêt, nous avons opté pour l'option HA".

Les architectes A4, A3 et A7 ont répondu : *"Une panne complète d'une zone est probablement très rare, donc l'option HA est importante, mais pas au point d'être indispensable"*. Nous avons posé plusieurs questions sur la sélection des services cloud de base de données relationnelles et nous nous sommes assurés que la liste des NFA précédemment collectée était pertinente pour notre étude.

Dans cette étude de cas, nous avons rassemblé les questions soulevées par l'étude présentée à la section 4.2.1. Ensuite, sur la base des plans des fournisseurs de services, nous avons identifié les NFA, les NFR et leurs relations qui répondent aux questions des architectes pour les services de bases de données relationnelles. Ensuite, nous avons passé en revue les précédents travaux de bernchmarks sur les services de bases de données relationnelles afin d'identifier les NFA qui influencent les performances. Enfin, nous avons mené une deuxième étude empirique avec des architectes sur les services de bases de données relationnelles. Cette étude a permis de s'assurer que la liste des NFA et des NFR est complète pour le processus de sélection des services de bases de données relationnelles. Par exemple, nous avons supprimé de la liste des NFA le nombre de serveurs sur lesquels le service est déployé. En effet, les architectes nous ont expliqué qu'ils ont demandé cette information à plusieurs fournisseurs de services dans le passé et que ces fournisseurs ne veulent pas divulguer cette information.

Dans cette étude, nous avons élaboré une liste pertinente de NFA et de leurs influences sur les NFR (comme indiqué dans le tableau 4.2). Cette liste sera utilisée dans la prochaine section (la section 5) afin de sélectionner les meilleurs services cloud de bases de données relationnelles en fonction des exigences des architectes en matière d'applications.

4.3.2 Étude de cas 2 : services cloud file de messagerie

Les services cloud file de messagerie permettent le découplage et la mise à l'échelle des micro-services, des systèmes décentralisés et des applications sans serveur. Ils permettent d'envoyer, de stocker et de recevoir des messages entre des composants logiciels [121].

De nombreux fournisseurs de services tels qu'Amazon [35], Azure [36] et Google [37] proposent des services cloud file de messagerie pour les entreprises, qui présentent des différences importantes. Les architectes ne savent pas sur quelle base évaluer ces services. Dans cette section, nous appliquons notre méthodologie (Section4.2) pour identifier les éléments clés de comparaison et leurs relations pour les services cloud file de messagerie.

4.3.2.1 Résultat de l'identification des besoins des architectes

Les résultats obtenus pour cette étude de cas sont les mêmes que ceux obtenus dans la sous-section 4.3.1.1. Les résultats de cette étude sont génériques et réutilisables pour tous les types de services.

4.3.2.2 Résultat de l'identification des attributs des plans de service et de leur influence sur les NFR

Nous allons détailler les attributs des services cloud file de messagerie. Un grand nombre de solutions sont disponibles pour ces services et sont mises en œuvre par différents fournisseurs de services cloud. Les attributs intéressants des services cloud file de messagerie sont les suivants.

Pour les services cloud file de messagerie, les seuls attributs de capacité qui peuvent être identifiés sont les suivants :

— Nombre de requêtes par mois
— Limite de mémoire (GB)
— Nombre de files d'attente
— Nombre de messages par file d'attente
— Limite de la taille des messages
— Nombre de connexions simultanées
— Taux d'envoi (mégabits/seconde)

Le principal attribut de capacité est bien sûr le degré d'envoi.

Les attributs fonctionnels des services cloud file de messagerie que nous pouvons identifier sont les suivants :

— Monitoring

35. https ://aws.amazon.com/fr/elasticache/
36. https ://azure.microsoft.com/en-us/services/cache/
37. https ://cloud.google.com/memorystore

— Audit

— Web-Stomp/Websockets - protocoles

— Tableau de bord

Les attributs de service sont classiques et sont plus ou moins toujours les mêmes :

— Support (support STOMP - The Simple Text Oriented Messaging Protocol, support MQTT, MQ Telemetry Transport, support SSL, etc).

— SLA.

L'influence des NFA sur les NFR pour les services cloud file de messagerie ne peut être déterminée que de deux façons : en utilisant l'expertise des architectes ou en consultant la documentation technique sur la question (sauf pour les performances pour lesquelles le temps de réponse et le débit sont suffisants pour l'évaluer pour le service). En raison de la vision très technique des architectes, il n'a pas été possible de recueillir des informations auprès d'eux. Nous avons consulté la documentation technique sur les services cloud file de messagerie [122] et leurs comparaisons avec les technologies de journalisation distribuées (ces technologies ajoutent de nouveaux types de solutions pour déplacer les données dans certains cas d'utilisation) [123] telles que Apache Kafka[38], Amazon Kinesis[39], Microsoft Event Hubs[40] et Google Pub/Sub[41]. Nous avons appris, par exemple, de [124] que les stratégies de gestion des partitions de réseau influencent la fiabilité selon trois modes :

— ignore - votre réseau est vraiment fiable. Tous vos nœuds sont dans un rack, connectés à un commutateur, et ce commutateur est aussi la route vers le monde extérieur. Vous ne voulez pas courir le risque qu'un de vos clusters se déconnecte si une autre partie de celui-ci tombe en panne (ou si vous avez un cluster à deux nœuds).

— pause_minority - votre réseau est peut-être moins fiable. Vous avez regroupé trois centres de données et vous supposez un seul centre de données tombera en panne à la fois.

— autoheal - votre réseau peut ne pas être fiable. Vous êtes plus préoccupé par la continuité du service que par l'intégrité des données. Vous avez peut-être une grappe de deux nœuds.

En raison de la difficulté à déterminer le degré d'influence des NFA sur les NFR (expliqué dans la section 4.2.2), nous présentons dans le tableau 4.3 les attributs techniques

38. https ://www.cloudkarafka.com/
39. https ://aws.amazon.com/fr/kinesis/
40. https ://azure.microsoft.com/en-us/services/event-hubs/
41. https ://cloud.google.com/pubsub/docs/overview

ayant l'impact positif, neutre ou négatif estimé des NFA (+,=,-). À ce stade, c'est tout ce que nous pouvons dire sur cet impact.

Attributs	Performance	Disponibilité	Fiabilité	Sécurité	Scalabilité
latency	+	=	=	=	=
throughput	+	=	=	=	=
Replication	-	+	+	=	=
Multi region	-	+	+	-	=
deal with network partition (mode ignore)	=	=	+	=	=
deal with network partition (pause minority)	=	=	+	=	=
deal with network partition (autoheal)	=	=	-	=	=
Shared instance	-	=	-	=	=
Dedicated instance	+	=	+	+	=
messaging guaranties (At least One)	-	=	+	=	=
messaging guaranties (precisely Once)	=	=	+	=	=
messaging guaranties (is not guaranteed)	+	=	-	=	=
Ack	-	=	+	=	=
VPC	=	=	=	+	=
Encryption At Rest	=	=	=	+	=
Persistence	-	=	+	-	=
Configurable persistence period	-	=	+	=	=
Scaling not supported	=	=	=	=	-
Scaling converted to other storage types	=	=	=	=	=
Scaling Auto-increase	=	=	=	=	+
Long Polling	+	=	=	=	=
Batching Policies	+	=	=	=	=

TABLE 4.3 – Influences des NFA sur les NFR pour les services cloud file de messagerie

Dans cette sous-section, nous avons répondu aux questions des architectes (soulevées dans la sous-section 4.3.1.1) en dressant une liste des NFA à partir des plans des fournisseurs de services cloud file de messagerie et en identifiant leurs influences (positives ou négatives) sur les différents NFA.

4.3.2.3 Résultat de l'identification des attributs sur la base de benchmarks

La plupart des fournisseurs de services cloud file de messagerie proposent le débit et le temps de réponse comme indicateurs de performance pour leurs services. Ces deux indicateurs de performance sont suffisants pour évaluer la performance des services cloud file de messagerie. Par conséquent, nous devons recueillir le temps de réponse et le débit des services pour les fournisseurs qui ne fournissent pas ces informations. À cette fin, nous

avons examiné des benchmarks qui comparent ces services avec ceux dont nous connaissons le temps de réponse et le débit [125]. Cela a permis de récupérer tous les temps de réponse et le débit de tous les services cloud file de messagerie des fournisseurs. Ainsi, nous utiliserons ces deux NFA pour évaluer la performance des plans des services.

4.3.2.4 Sélection des NFA et des NFR à partir d'une étude empirique

Nous avons réalisé une étude empirique avec des architectes sur les services cloud file de messagerie et recueilli plusieurs réponses. Par exemple, à la question "comment justifiez-vous le choix du service cloud de file de messagerie actuellement utilisé dans votre application ?". Tous les architectes ont répondu : *"Notre choix était basé sur les exigences de l'application, nous avions besoin d'un service qui garantit la livraison des messages au moins une fois et des ordres en mode FIFO. Nous avions besoin d'un service capable de traiter un maximum de messages par minute et avec une taille de message de 256 Ko ou plus. Nous avions besoin d'un service offrant une persistance des messages avec une période de rétention supérieure à 4 jours pour récupérer les messages si nécessaire. Nous avions besoin d'un service file de messagerie qui ne prend en charge qu'un seul consommateur à la fois. Enfin, les coûts généraux d'exploitation du service doivent être réduits au maximum. Toutes ces considérations ont guidé notre choix du service cloud file de messagerie"*. Nous avons posé plusieurs questions similaires et nous nous sommes assurés que la liste des NFA et de leurs influences sur les NFR (que nous avions recueillies lors de nos précédentes études) était complète et pertinente pour le choix des services cloud file de messagerie.

Dans cette étude de cas, nous avons réutilisé les questions tirées de l'étude 4.3.1.1 pour recueillir les NFA des plans de service file de messagerie et leurs relations. Par exemple, nous avons identifié les NFA : type d'instance, garanties de messagerie, cryptage des données et Ack. Parmi les exemples de l'influence des NFA sur les NFR, on peut citer le Ack, qui a une influence positive sur la fiabilité et une influence négative sur les performances. Ensuite, nous avons examiné les travaux de benchmarks sur les services file de messagerie où nous avons recueilli des informations indiquant quels NFA influencent les performances. Nous avons utilisé le débit et le temps de réponse pour évaluer les performances de ces services. Enfin, nous avons mené une deuxième étude empirique avec les architectes pour nous assurer que la liste des NFA et des NFR est complète pour le processus de sélection des services file de messagerie. Par exemple, nous avons supprimé le monitoring de la

liste des NFA. En effet, Les architectes nous ont expliqué que la plupart des paramètres de monitoring ne sont pas disponibles pour les services et que, dans tous les cas, le déploiement d'un service de monitoring est essentiel pour suivre la santé de leurs applications.

Notre étude a permis de dresser une liste intéressante de NFA et de leurs influences sur les NFR (comme le montre le tableau 4.3) afin de sélectionner les services cloud file de messagerie en fonction des exigences des architectes en matière d'application.

4.3.3 Étude de cas 3 : services cloud de gestion de cache

Un service cloud de gestion de cache est un service qui facilite la configuration, l'exécution et le passage à l'échelle des backups de données en mémoire. Ce service améliore les performances des applications web en permettant de récupérer des données à partir des backups de données en mémoire de manière rapide. Ainsi, les performances des applications web ne dépendent plus uniquement des bases de données sur disque, qui sont plus lentes. Plusieurs fournisseurs de services tels que Heroku [42], Azure [43], Google [44] et Amazon [45] proposent des services cloud de gestion de cache pour les entreprises, avec des configurations très différentes. Les architectes ne savent pas sur quelle base évaluer ces services. Dans cette section, nous appliquons notre méthodologie (Section4.2) pour identifier les éléments clés de comparaison et leurs relations pour les services cloud de gestion de cache.

4.3.3.1 Résultat de l'identification des besoins des architectes

Les résultats obtenus pour cette étude de cas sont les mêmes que ceux obtenus dans la sous-section 4.3.1.1. Les résultats de cette étude sont génériques et réutilisables pour tous les types de services.

4.3.3.2 Résultat de l'identification des attributs des plans de service et de leur influence sur les NFR

Pour valider notre méthodologie, nous détaillons les attributs des services cloud de gestion de cache. Ces services sont proposés par différents fournisseurs de services cloud. Les attributs ci-dessous sont les attributs pertinents des services cloud de gestion de cache :

— la quantité de RAM en GB

42. https ://elements.heroku.com/addons
43. https ://azure.microsoft.com/en-us/services/cache/
44. https ://cloud.google.com/memorystore
45. https ://aws.amazon.com/fr/caching/aws-caching/

— le nombre de nœuds ou de conteneurs ou de caches

— le nombre de requêtes par mois

— le nombre maximum de connexions utilisateurs

— le débit exprimées en Mégabits par seconde (Mbit/s)

— le nombre de Ko par seconde (KB/s)

Les attributs fonctionnels des services cloud de gestion de cache que nous pouvons identifier sont les suivants :

— Monitoring

— Audit

— Récupération des journaux

— Tableau de bord

Les attributs des services sont standard et restent essentiellement les mêmes :

— Support SSL/TLS

— SLA

L'influence des NFA sur les NFR pour les services cloud de gestion de cache peut être déterminée de deux manières, soit en utilisant l'expertise des architectes, soit en consultant la documentation technique sur le sujet [126] [127] [128](sauf pour les performances pour lesquelles un benchmarks supplémentaire peut être nécessaire). Nous avons appris, par exemple, que les requêtes non-SSL sont plus rapides que les requêtes SSL. Nous avons également appris que la détection automatique des pannes et la réplication en lecture seule augmentent la disponibilité. En effet, la détection automatique d'un nœud maître défectueux entraîne une procédure de basculement vers l'un des réplicas en lecture seule.

En raison de la difficulté à déterminer le niveau d'influence des NFA sur les NFR (expliqué dans la section 4.2.2), nous présentons dans le tableau 4.4 les attributs techniques ayant l'impact positif, neutre ou négatif estimé des NFA(+,=,-). À ce stade, c'est tout ce que nous pouvons dire sur cet impact.

Dans cette sous-section, nous avons répondu aux questions des architectes (soulevées dans la sous-section 4.4) en dressant une liste des NFA à partir des plans des fournisseurs de services cloud de gestion de cache et en identifiant leurs influences (neutre, positives ou négatives) sur les différents NFR.

Attributs	Performance	Disponibilité	Fiabilité	Sécurité	Scalabilité
Vcpu	+	=	=	=	=
Bandwidth	+	=	=	=	=
Throughput	+	=	=	=	=
RAM	+	=	=	=	=
IOPS	+	=	=	=	=
Data persistence (Backup)	-	+	+	=	=
Replication and Failover	-	+	+	=	=
Shared instance	-	=	-	=	=
Dedicated instance	+	=	+	+	=
Encryption at rest	=	=	=	+	=
Encryption in transit	=	=	=	+	=
VPC	=	=	=	+	=
Multi region	-	+	+	=	=
Number of read replicas	+	=	+	=	+
Horizontal scalability	=	=	=	=	+
Vertical scalability (Change node type)	=	=	=	=	+
Automatic failure detection	=	=	+	=	=
Number of nodes or containers	+	=	+	-	=

TABLE 4.4 – Influences des NFA sur les NFR pour les services cloud de gestion de cache

4.3.3.3 Résultat de l'identification des attributs sur la base de benchmarks

Dans la littérature existante, peu de travaux ont été réalisés sur les benchmarks des services cloud de gestion de cache ou sur la comparaison des performances de ces services entre différents fournisseurs. Par conséquent, nous avons utilisé la documentation technique sur les résultats des évaluations des performances des services cloud de gestion de cache [129] [130] (benchmarks du même fournisseur) pour comprendre quels attributs influencent les performances de ces services. Les services de cache s'exécutent sur les machines virtuelles des fournisseurs de services. Ces machines ont un impact direct sur la performance de ces services. Par conséquent, les NFA qui influencent les performances des machines virtuelles où le service est déployé influencent également les performances des services cloud de gestion de cache. Parmi les exemples, citons le modèle de CPU, l'hétérogénéité matérielle et le modèle de tenant (les mêmes attributs que ceux décrits dans le cas d'utilisation 4.3.1.2, pour l'influence des instances sur les services de bases de données relationnelles). Cependant, le degré d'influence des instances sur la performance des services diffère d'un service à l'autre.

Pour évaluer la performance des services cloud de gestion de cache, il existe une série d'indicateurs de performance, tels que la vitesse à laquelle le service traite une requête de l'utilisateur, la quantité de données qu'un service est capable de traiter pour chaque requête, le nombre de requêtes traitées par le service dans un intervalle donné, ou le temps d'attente de l'utilisateur pour obtenir une réponse à sa requête. Sur le plan technique, ces indicateurs de performance sont exprimés en termes de débit (ou bande passante) et de latence. Ces deux facteurs sont très difficiles à obtenir car ils dépendent de l'architecture sur laquelle le fournisseur de services souhaite mener ces expériences et des conditions dans lesquelles il envisage de les réaliser. Par exemple, Azure [131] propose d'évaluer les performances de ses services cloud de gestion de cache en calculant le nombre de requêtes non-SSL ou SSL par seconde (RPS) sur une base de 1Ko. Il est très difficile de trouver un autre fournisseur qui calcule le débit dans les mêmes conditions et avec la même approche.

Dans cette sous-section, nous avons collecté la liste des attributs qui influencent la performance des services cloud de gestion de cache.

4.3.3.4 Résultat de la sélection des NFA et des NFR à partir d'une étude empirique

Nous avons mené une étude empirique avec les architectes sur les services cloud de gestion de cache, comme mentionné dans la sous-section 4.2.4, Par exemple, à la question "comment justifiez-vous le choix du service cloud de gestion de cache actuellement utilisé dans votre application". Tous les architectes ont répondu : *"Nous avons choisi ce service parce qu'il nous permette de détecter automatiquement les pannes et de passer à une réplique en lecture seule, augmentant ainsi la disponibilité de notre application sans aucune intervention manuelle de notre part. De plus, ce service offre une réplication multi-AZ de nos caches, ce qui garantit une meilleure fiabilité de notre service. En outre, comme les problèmes d'infrastructure sous-jacents peuvent entraîner des pertes de données, nous avions besoin d'un service qui nous permettait de conserver les données du cache pour notre application (persistance des données). En ce qui concerne la sécurité, nous avions besoin d'un service qui nous permettait de chiffrer toutes les communications entre les clients et le serveur Redis ainsi qu'entre les serveurs Redis (nœuds primaires et nœuds de lecture répliqués). Par ailleurs, nous avions besoin d'isoler nos ressources en définissant des plages d'adresses IP que nous voulions utiliser pour nos nœuds, tout en établissant la connexion*

à d'autres applications (isolation du réseau ou VPC). Pour la mise à l'échelle, nous avions besoin de la flexibilité nécessaire pour ajouter ou supprimer des nœuds de cache (mise à l'échelle horizontale). Enfin, les frais généraux liés à l'exploitation du service devaient être aussi faibles que possible. Toutes ces considérations ont guidé notre choix du service cloud de gestion de cache".

Dans cette étude de cas, nous avons dressé une liste pertinente des NFA et de leurs influences sur les NFR (comme le montre le tableau 4.3) afin de sélectionner les services cloud de gestion de cache en fonction des exigences des architectes en matière d'application.

4.3.4 Conclusion

Dans cette section, nous avons proposé une méthodologie pour identifier les éléments clés de comparaison et leur relation pour la sélection des services cloud. Elle est basée sur des données réelles disponibles auprès des fournisseurs de services et des benchmarks. Tout d'abord, nous avons mené une enquête auprès des architectes afin de comprendre leurs exigences pour la sélection de services cloud. Ensuite, nous avons identifié les questions qui intéressent les architectes et nous les avons classées en fonction de leur influence sur les NFR. Ensuite, nous avons identifié les attributs des plans des fournisseurs de services qui répondent aux questions des architectes et leur influence sur les NFR à partir de la documentation technique. En outre, nous avons examiné les travaux sur les benchmarks des services cloud afin de recueillir les attributs qui influencent les performances. Enfin, nous avons mené une étude empirique pour nous assurer que la liste des attributs que nous avons identifiés était complète et pour ajouter ceux qui manquaient. Nous considérons notre travail comme une première tentative pour constituer la base d'un processus décisionnel solide qui fait défaut aujourd'hui.

La méthode que nous proposons peut être réutilisée pour tout type de service. Toutefois, elle reste dépendante de la disponibilité des données (NFA) communiquées par les fournisseurs de services et des tests de benchmarks disponibles dans la littérature. Notre méthode doit être appliquée pour chaque type de service afin de collecter une liste pertinente des NFA et leurs influences sur les NFR. Cette liste de NFA est spécifique pour chaque type de service. Afin d'améliorer la méthode proposée, les perspectives suivantes peuvent être envisagées : **(i)** demander aux fournisseurs de services de communiquer des mesures universelles à des fins de comparaison. Par exemple, leur demander de fournir le

débit et la latence de leurs plans de services selon les mêmes scénarios expérimentaux et sous les mêmes contraintes. Ainsi, il serait possible d'évaluer les performances des plans des services. **(ii)** Élargir les discussions avec les architectes pour obtenir plus d'informations sur les NFA et leurs influence sur les NFR. **(iii)** Réappliquer notre méthode pour différents types de services tels que les services d'apprentissage automatique.

Dans les prochains chapitres, nous allons utiliser les résultats de cette étude pour évaluer les services cloud et la composition des services cloud. Tout d'abord, l'importance relative des attributs et des NFR devrait être déterminée sur la base des types d'application et l'exactitude des résultats devrait être validée par les architectes. Ensuite, sur la base du résultat précédent, une méthodologie sera proposée pour évaluer les services cloud et la composition des services cloud. Le travail présenté dans ce chaptire est un premier pas vers la sélection guidée d'un ensemble de services qui répondent aux exigences d'une architecture complète.

Chapitre 5

Sélection de services cloud

5.1 Introduction

Dans le chapitre précédent, nous avons abordé la première problématique qui consiste à identifier les éléments clés de comparaison et leur relation pour la sélection des services cloud. À cet égard, nous avons proposé une méthode basée sur des études empiriques avec des architectes, la collecte des NFA à partir des informations provenant des fournisseurs de services cloud, des travaux antérieurs sur les benchmarks des services cloud et des études qui effectuent une revue de la littérature à grande échelle pour collecter et codifier les recherches existantes sur la prévisibilité des performances des services cloud. Nous allons nous intéresser dans le présent chapitre à la sélection des services cloud.

Poussées par la nécessité de réduire les coûts de production et d'améliorer l'ensemble de leurs activités, les entreprises informatiques utilisent des services cloud. Ces services permettent de bénéficier d'un développement d'applications simple et rapide. Un architecte logiciel est capable de développer et de déployer une application entière par le biais de services cloud en utilisant de nombreux services disponibles sur le marché. Toutefois, le principal défi, à cet égard, est la sélection des services appropriés en fonction d'un ensemble d'attributs non fonctionnels (NFA). La sélection des services cloud appropriés est un sujet de préoccupation majeur [8] [132] [16] [17]. Toutefois, les travaux existants présentent des limites, soit en se concentrant uniquement sur les performances et le prix sans tenir compte d'autres NFR telles que la sécurité et la scalabilité, soit en utilisant des attributs tels que le degré de disponibilité et le temps d'exécution, qui ne sont pas disponibles dans des cas réels.

Compte tenu du nombre important de fournisseurs de services proposant des services fonctionnellement similaires, le défi pour les architectes consiste à sélectionner le meilleur service qui répond le mieux à leurs besoins non fonctionnels. Cependant, le processus de sélection des services cloud n'est pas trivial. Plusieurs facteurs contribuent à la complexité du processus. Premièrement, le niveau de connaissance et de compréhension des exigences des services cloud varie considérablement en fonction de l'application que les architectes veulent déployer. Deuxièmement, les services cloud sont décrits par différents NFA et leur influence sur les NFR varie en fonction des types de services et des besoins applicatifs des architectes. Pour évaluer les services, nous devons comprendre l'impact de ces NFA sur les NFR des services en fonction des besoins applicatifs des architectes. Par ailleurs, comme la performance est un NFR particulier, nous devons déterminer le degré d'influence des NFA sur la performance en nous basant sur les benchmarks et l'analyse de stabilité de la performance fournis par les études sur la variabilité de la performance des services cloud. Troisièmement, les services évalués peuvent avoir d'excellentes valeurs pour certains NFR et de très mauvaises valeurs pour d'autres. Une telle situation doit être évitée afin de fournir aux architectes les meilleurs services pour leurs besoins avec des exigences minimales sur tous les NFR de leur service.

Ainsi, notre objectif est d'aider les architectes à sélectionner les meilleurs services en fonction de leurs besoins en utilisant différents scénarios d'utilisation. Pour ce faire, nous proposons, sur la base des résultats obtenus au chapitre précédent, une méthodologie permettant de comparer des services ayant des exigences fonctionnelles similaires.

Ce chapitre est organisé comme suit : la section 2 introduit les définitions formelles pour modéliser notre problématique. La section 3 décrit la méthodologie proposée pour la sélection des services cloud. Avant de conclure ce chapitre, la section 4 présente la validation de notre méthodologie sur les services cloud SQL, les services cloud de mise en cache et les services cloud de file d'attente.

5.2 Formalisation de la sélection des services

Dans cette section nous introduisons les concepts de base de la sélection des services cloud et leurs definitions formelles.

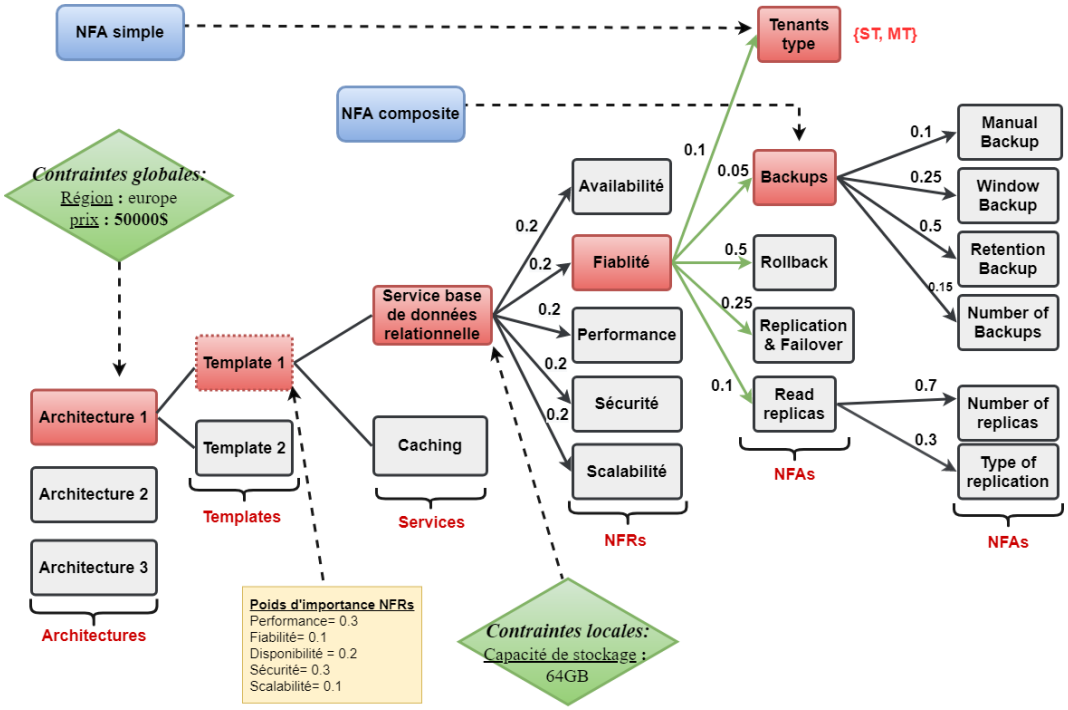

FIGURE 5.1 – Exemple NFR, NFA simple et NFA composite

5.2.1 Définition d'un NFA simple chaine de caractères

Soit A l'ensemble de tous les NFA existants. Un $NFA_{s_c} \in A$ est défini comme suit : $NFA_{s_c} = (l, \{c_i\} | i \in [1, n])$ avec :

— l : dénote le label associé au NFA chaine de caractères.

— $\{c_i\}$: l'ensemble de valeurs associées au NFA de type chaine de caractère avec $c_i \in \Sigma$ (l'ensemble des chaines de caractères).

Exemple : Un exemple d'un NFA simple de type chaine de caractères est présenté dans la figure 5.1 :

l= "Tenants Type" : $c_1 = ST$ et $c_2 = MT$ avec ST : Single Tenant et MT : Multi-tenant.

5.2.2 Définition d'un NFA simple numérique

Soit A l'ensemble de tous les NFA existants. Un $NFA_{s_n} \in A$ est défini comme suit : $NFA_{s_n} = (l, \{v_i\} | i \in [1, n])$ avec :

— l : dénote le label associé au NFA numérique.

— $\{v_i\}$: l'ensemble de valeurs associées au NFA numérique avec $v_i \in \mathbb{D}$

5.2.3 Définition d'un NFA composite

Un NFA composite est un arbre n-aire pondéré que nous notons A_{NFA}. Soit N l'ensemble de noeuds. A_{NFA} est défini récursivement comme suit :

— Si $n \in N$, et si $A_h = \{a_{i_{nfa}} \mid i \in [1,h]\}$ un ensemble des arbres n-aires, avec h est la cardinalité de A_h , alors ((n,$\{a_{i_{nfa}}\}$),L_w) est aussi un arbre n-aire avec :

— n est le nœud père de chacun des $a_{i_{nfa}}$.

— $a_{i_{nfa}}$ est un arbre n-aire et fils du nœud n.

— $L_w : A_h \rightarrow [0,1]$: Fonction qui associe à chaque fils du noeud n un poids w_i dans $[0,1]$.

— le noeud n doit satisfaire la contrainte suivante : $\sum_{i=1}^{h} L_w(w_i) = 1$

Exemple (voir la figure 5.1)

Comme le montre la figure 5.1, le NFA Backups est composite :

N= (Backups, Manual Backup, Window Backup, Retention Backup, Number of Backups).

$A_{Backups}$ est défini comme suit :

— n : Backups est le noeud père.

— h= 4 (nombre des noeuds fils)

— L'ensemble des arbres n-aires et fils de n sont :

($a_{1_{Backups}}$) : (Manual Backup)

($a_{2_{Backups}}$) :(Window Backup)

($a_{3_{Backups}}$) : (Retention Backup)

($a_{4_{Backups}}$) : (Number of Backups)

— Le noeud Backups satisfait la contrainte suivante : $\sum_{i=1}^{4} L_w(e_i) = 0.1+0.15+0.5+0.25=1$

5.2.4 Définition des Contraintes locales

Soit C l'ensemble de toutes les contraintes sur l'ensemble des services. L'ensemble de contraintes locales $C_l \subset C$ sur un service est défini comme suit : $C_l = (\{c_{l_k}\}, k \in [1,t])$ avec :

— c_{l_k} est la $k^{\text{ième}}$ contrainte locale.

Exemple : comme le montre la figure 5.1 une contrainte locale peut être par exemple "région :europe" et "Capacité de stockage $\geq$ 64Gb".

5.2.5 Définition d'un service

Soit S l'ensemble de tous les services existants. Soit $s \in S$ un service défini comme suit : $s = (\{NFA_i\}, C_l \mid , i \in [1, m])$, m est le nombre des NFA dans le service s, avec :

— $\{NFA_i\}$ est l'ensemble d'attributs non fonctionnels (voir les définitions 5.2.2, 5.2.1 et 5.2.3).

— C_l est l'ensemble des contraintes locales sur s (voir la définition 5.2.4).

Exemple

Dans la figure 5.1 : s est un service de base de données relationnelle défini par :NFA_1=Tenants type, NFA_2=Backups, NFA_3=Rollback, NFA_4=Replication Failover, NFA_5=Read replicas ; C_l={région :europe, Capacité de stockage $\geq$ 64Gb}

5.2.6 Définition d'un arbre de besoins NFR

Soit V_{nfr}=($performance$, $sécurité$, $disponibilité$, $scalabilité$, $fiabilité$) l'ensemble des NFR. Soit N_{nfr} l'ensemble des noeuds. Un $NFR \in V_{nfr}$ est un arbre n-aire pondéré défini récursivement comme suit :

$NFR = ((n, \{A_{NFA_i}\}), L_{w_n})$ avec $i \in [1, p]$ et p est la cardinalité de $\{A_{NFA_i}\}$:

— n est le nœud père de chacun des A_{NFA_i}

— A_{NFA_i} est un arbre n-aire qui présente un NFA et est le fils du nœud n (voir la définition 5.2.3).

— $L_{w_n} : \{A_{NFA_i}\} \rightarrow [0, 1]$: Fonction qui associe à chaque élément A_{NFA_i} un poids w_i dans $[0, 1]$.

— le noeud n doit satisfaire la contrainte suivante : $\sum_{i=1}^{p} L_{w_n}(w_i)$=1.

Exemple :

On considère le NFR fiabilité (5ème NFR pour le service "s" de base de données relationnelle). Comme le montre la figure 5.2, ce NFR est défini comme suit :

— n : Fiabilité est le noeud père.

— A_{NFA_1} :(Tenant Type)

A_{NFA_2} :(Rollback)

A_{NFA_3} :(Backups, (Manual Backup, Window Backup, Retention Backup, Number of backups per day))

A_{NFA_4} : (Replication & failover)

A_{NFA_5} : (Read replicas, (Number of replicas, Type of Replication))

— le noeud Fiabilité satisfait la contrainte suivante : $\sum_{i=1}^{5} L_w(w_i) = 0.1+0.05+0.5+0.25+0.1=1$

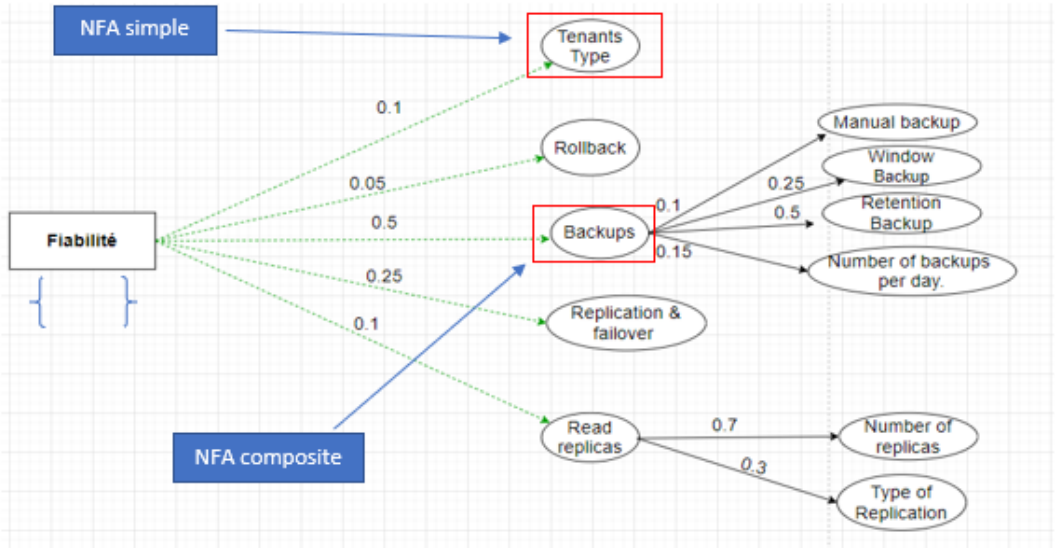

FIGURE 5.2 – Exemple NFR/NFA

5.2.7 Besoins des architectes en termes de NFR pour un service

soit $s \in S$ un service et soit V_{nfr} le vecteur NFR de dimension k.

— $W_s = (w_1, .., w_j, .., w_k)$: est un vecteur de pondération, pour le service s, de dimension k tel que :

— $w_j \in [0, 1]$.

— $\sum_{j=1}^{k} w_j = 1$

w_j est donnée par les architectes et dénote le poids d'importance d'un NFR.

Exemple Dans la figure 5.1 : Pour le service de base de données relationnelle, V_{nfr}= ($Performance$, $Fiabilité$, $Disponibilité$, $Sécurité$, $Scalabilité$)

Le vecteur de pondération pour l'ensemble de NFR est respectivement comme suit :

$W_s = (0.2, 0.2, 0.2, 0.2, 0.2)$.

5.2.8 Définition d'un plan d'un service

Soit $s \in S$ un service et $p_k \in P$ le k ième plan dans le service s. $p_k \in P$ est défini comme suit :

$p_k = (F_{(p)}, C_p)$

— $F_{(p)} : P \rightarrow \mathbb{D}$ une fonction qui associe à chaque NFA dans le plan p_k une valeur dans D.

— $C_p : P \rightarrow \mathbb{R}+$ est une fonction qui associe à chaque plan un coût.

5.2.9 Le vecteur de valeurs de NFR pour un plan de service

Soit $p \in P$ un plan de service $s \in S$. V_{nfr} le vecteur des NFR de dimensions k.

Soit V_p le vecteur de valeurs de NFR de dimension k pour le plan p de s. Ce vecteur est calculé en utilisant l'algorithme 1.

Un exemple de ce vecteur est présenté dans le tableau 5.1. Soit $s \in S$ un service de base de données et soit $p \in P$ un plan "ServiceSQL : Amazon-db.r3.4xlarge" de s. Le vecteur $V_p = (1.0, 0.057, 0.56, 1.0, 0.926)$, avec Disponibilité= 1.0, Fiabilité=0.057, Performance=0.56, Scalabilité =0.56, et Sécurité=0.926.

Service obtenu	Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	Prix de services
ServiceSQL : Amazon - db.r3.4xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.560	1.0	0.926	3291 $
ServiceSQL : Amazon - db.r4.4xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.553	1.0	0.926	3339 $
ServiceSQL : Amazon - db.r3.8xlarge (Provisioned IOPS - monoAZ)	0.4819925	0.899	0.879	1.0	0.939	3167 $
cloudamqp- RabbitMQ - Power Panda - 3 nodes - multiAZ	1.0	0.855	0.402	0.5	0.737	499 $
FileAttente1 :SQS Standard	1.0	0.988	0.155	1.0	0.737	4 $
FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $
ServiceSQL : Amazon - db.r3.4xlarge (General Purpose - multiAZ)	1.0	0.957	0.48	1.0	0.926	3059 $
Service SQL : Amazon - db.r4.8xlarge (Provisioned IOPS - monoAZ)	0.481	0.899	0.866	1.0	0.939	3215 $

TABLE 5.1 – La configuration des plans de services

5.3 Méthode proposée

Comme indiqué dans la sous-section 1.1, il est possible de construire et de développer une architecture complexe en combinant uniquement des services de différents fournisseurs de cloud. C'est une forme de composition de services qui permet la mise en production des applications [133].

Dans cette section, nous proposons une méthode pour sélectionner les plans de service les plus appropriés pour une priorité de besoins en considérant les caractéristiques des ser-

vices disponibles. Cette méthode comprend **(1)** la réalisation d'une enquête pour attribuer des poids significatifs aux NFA et aux NFR en fonction des besoins des architectes de l'application considérée ; **(2)** la résolution des difficultés liées à l'incapacité de mesurer l'impact des NFA sur la performance ; **(3)** la proposition d'un modèle de sélection de service. Ce modèle est basé sur : **(i)** la normalisation des valeurs des NFA en fonction des besoins applicatifs des architectes. Les méthodes de normalisation que nous proposons dépendent des types de données des NFA ; **(ii)** l'application de la méthode d'aide à la décision OWA *(Ordered weighted averaging aggregation operator) [134]* pour calculer (agréger) les valeurs NFA et NFR ; et **(iii)** la proposition d'un algorithme pour l'évaluation des services cloud en fonction des besoins des architectes. Cet algorithme utilise les résultats des valeurs de normalisation NFA et la méthode OWA pour calculer un score global pour chaque plan de service. Le résultat de cet algorithme est la proposition d'une liste de plans de services alternatifs qui répondent aux besoins des architectes de telle sorte que le coût soit inférieur à un prix défini par ces derniers.

5.3.1 Enquête pour l'attribution de poids significatifs NFA /NFR

Dans le chapitre 4, nous avons identifié les NFA nécessaires pour évaluer les services cloud et leurs influences (positives, négatives ou neutres) sur les NFR. Ces NFA et leurs influences sur les NFR varient selon le type de services. Cependant, pour évaluer les services, nous devons mieux comprendre l'impact de ces NFA sur les NFR des services. À cette fin, nous avons interrogé sept architectes pour comprendre leurs points de vue sur ce sujet. Ces architectes ont été choisis en fonction de leur expérience dans le management des services cloud. Ces architectes avaient tous au moins sept ans d'expérience. Dans cette perspective, nous avons établi un système d'évaluation comportant un total de 100 points. Les architectes doivent ainsi partager 100 points sur les différents NFA qui influencent chaque NFR et partager 100 points sur les NFR qui influencent chaque service. Par exemple, la fiabilité des services de bases de données relationnelles est influencée par les NFA suivants : sauvegardes, type de tenant, type de déploiement, répliques de lecture et rollback. Les architectes sont invités à noter chacun de ces NFA (selon les degrés d'influence de ces NFA sur la fiabilité de leur application), de sorte que la somme de leurs notes soit égale à 100 points. La même méthode s'applique aux influences des NFR sur le service. Les architectes doivent, en fonction des exigences de leur application, distribuer 100 points sur le degré

d'influence des NFR (fiabilité, sécurité, etc.) sur le service. Ce système de points permet de s'assurer que les architectes étudient attentivement leurs besoins et contrôlent l'importance des NFA et des NFR pour les différents services.

Dans cette étude, nous avons constaté que les architectes étaient en mesure de répondre à toutes les questions relatives aux NFR, à l'exception de celles liées à la performance. Cette constatation nous a incités à réaliser une étude plus approfondie sur ce sujet, qui fera l'objet de la prochaine sous-section.

5.3.2 Évaluation de l'impact des NFA sur les performances

La plupart des travaux de recherche actuels utilisent des benchmarks pour évaluer la performance des services cloud. Ces méthodes sont coûteuses, longues à mettre en œuvre, valables durant une période de temps déterminée et pour des scénarios spécifiques. Par exemple, pour évaluer la performance des services cloud SQL, nous devons d'abord déployer ces services. Ensuite, nous devons établir des benchmarks pour chacun d'entre eux en fonction de différentes capacités de stockage. Par exemple, le coût d'un seul déploiement du service Heroku SQL de Citus Data[46] pour une capacité de stockage de 2 To s'élève à 9000 dollars. Avec des centaines d'autres services disponibles, le coût des tests augmentera considérablement. Il convient également de noter que les performances des services ne sont pas linéaires. En effet, le nombre de serveurs peut augmenter au fur et à mesure que l'on passe d'une capacité de stockage à une autre. Cela affectera considérablement les performances des services. Par conséquent, nous avons interrogé les architectes pour comprendre l'évolution des performances et comment chaque NFA affecte les performances des différents services. Les architectes ont tous répondu de la même manière : *"pour nous, la seule façon d'évaluer les performances est de se référer aux benchmarks"*. Cette réponse nous a incités à rechercher des indicateurs de référence pour évaluer les performances des services cloud.

Les indicateurs de référence pour évaluer les performances des services varient selon le type de service. Par exemple, pour les services SQL, les fournisseurs de services Amazon [135] et Azure [136] fournissent des informations sur les configurations de déploiement de leurs services, telles que le nombre de processeurs utilisés pour leurs instances. Le facteur d'influence du nombre de processeurs sur les performances n'est pas connu. En effet,

46. https://elements.heroku.com/addons/citus

cette influence dépend des caractéristiques des processeurs telles que leur type (Intel Xeon Platinum, AMD,...), le nombre de cœurs et leur puissance (exprimée en GHz). Pour permettre aux développeurs de comparer facilement la capacité de calcul des processeurs entre différents types d'instances, Amazon fournit des unités de calcul élastiques (ECU : Elastic Compute Units). Nous avons utilisé cette unité de calcul ainsi que les benchmarks [137] pour comparer la performance des instances d'autres fournisseurs avec celle d'Amazon. Ainsi, nous avons pu déterminer le nombre d'unités de calcul à attribuer aux instances d'autres fournisseurs.

Pour les services cloud de file d'attente et les services cloud de mise en cache, nous avons utilisé le débit et le temps de réponse comme indicateurs de performance pour évaluer la performance des services de cloud. Une explication plus détaillée de l'évaluation de la performance de chaque service est fournie dans la section validation 5.4.

Nous présentons dans la prochaine section des méthodes de normalisation permettant de normaliser les valeurs NFA en fonction de la nature de chacune d'entre elles.

5.3.3 Proposition d'un modèle de sélection de service

Dans cette section, nous présentons notre approche de la sélection des services cloud qui est basée sur : **(1)** la proposition de méthodes de normalisation des valeurs NFA en fonction de leurs types de données **(2)** l'utilisation de la méthode d'aide à la décision *Ordered weighted averaging aggregation operator (OWA)* pour classer les services cloud en fonction des besoins des architectes. À notre connaissance, cette méthode est la seule parmi les méthodes d'aide à la décision multicritères (MCDM) qui prend en compte à la fois les poids d'importance exprimés par les architectes et le compromis entre les valeurs les plus et les moins satisfaisantes. De cette façon, nous éliminons les solutions qui peuvent avoir des valeurs très satisfaisantes pour certaines NFA ou NFR et des valeurs insatisfaisantes pour d'autres. **(3)** la proposition d'un algorithme de sélection des services qui utilise les valeurs NFA normalisées et la méthode de prise de décision OWA.

5.3.3.1 Méthodes de normalisation des valeurs des NFA

La normalisation des données est cruciale pour toutes sortes de problèmes de prise de décision. Comme un grand nombre de NFA sont disponibles avec un très grand écart entre leurs valeurs, le problème d'avoir des valeurs normalisées infiniment petites peut se

poser. Ces valeurs normalisées peuvent fausser le résultat de l'évaluation des services. Par conséquent, le choix de méthodes de normalisation appropriées constitue un défi.

Cette section traite le problème de l'incompatibilité des mesures unitaires des différentes valeurs de NFA qui peuvent affecter les résultats de l'évaluation des services. Pour surmonter ce problème et pour exprimer les résultats en unités significatives et cohérentes, nous normalisons les valeurs des NFA entre 1 et -1 avant de commencer à évaluer les services cloud. Les valeurs normalisées des NFA doivent refléter la différence entre leurs valeurs sans les négliger. Par exemple, pour les services dont les prix des plans varient entre 9 et 10 000 dollars, la normalisation du prix de 9 dollars donne comme résultat une valeur si faible qu'elle pourrait être négligeable. Pour surmonter ce problème, plusieurs expérimentations ont été menées et, sur la base de la précision des résultats, nous proposons des méthodes avec les paramètres de normalisation suivants :

— $x = (x_1, ..., x_n)$ est l'ensemble de valeurs spécifiées pour le NFA_k du service.

— $min(x)$ est la plus petite valeur de l'ensemble x de telle sorte que $min(x) > value_j$, où $value_j$ est une valeur minimale prédéfinie pour un NFA_k. Cette valeur est spécifiée par les architectes en fonction de leurs besoins applicatifs. Par exemple, les architectes doivent conserver leurs sauvegardes pendant au moins trois jours. Dans ce cas, $min(x)$=3 pour la conservation des sauvegardes.

— $max(x)$ est la valeur la plus élevée de l'ensemble x tel que $max(x) < value_j$, où $value_j$ est la valeur maximale que peut prendre un NFA_k. Cette valeur est spécifiée par les architectes en fonction de leurs besoins applicatifs. Par exemple, les architectes ont besoin d'une rétention des sauvegardes de 7 jours pour leurs services SQL pour une application mobile. Pour eux, une rétention de plus de 7 jours n'est pas nécessaire. Par conséquent, la normalisation de la "rétention des sauvegardes" du NFA "Backups" doit être faite avec une valeur maximale de 7. Donc, $max(x)$=7 pour la "rétention des sauvegardes".

— x_i est la i^{me} valeur de l'ensemble x.

— z_i est la valeur normalisée de x_i.

Nous proposons des formules pour normaliser les valeurs des NFA en fonction des contraintes suivantes :

— **C1** : une valeur maximale est meilleure pour le service

— **C2** : une valeur maximale est mauvaise pour le service

— **C3** : $(min(x)/max(x)) \geq 0.01$ [47]

— **C4** : $(min(x)/max(x)) \leq 0.01$ [48]

1. Dans le cas où les contraintes C1 et C3 sont respectées, La formule (5.1) est extrêmement utile pour la comparaison relative entre les alternatives, c'est-à-dire que tout critère normalisé fournit la distance par rapport au meilleur concurrent [138].

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{5.1}$$

Exemple Plusieurs fournisseurs de services communiquent des informations sur le nombre d'IOPS utilisés pour leurs services. L'IOPS est une mesure de performance d'entrée/sortie utilisée pour caractériser les instances des services déployés. Plus la valeur des IOPS soit élevée, plus les performances du service sont élevées. Supposons que la valeur IOPS la plus faible est 128 et la plus élevée est 1000. Dans ce cas, $(min(x)/max(x)) = (128/1000) \geq 0.01$. Pour normaliser les valeurs IOPS, nous utiliserons la formule 5.1.

Le résultat de la normalisation d'une valeur IOPS de 192 est le suivant :

$$z_i = \frac{192-128}{1000-128} = 0.07339449541$$

2. Dans le cas où les contraintes C2 et C3 sont respectées, la formule (5.2), est extrêmement utile pour la comparaison relative entre les alternatives, c'est-à-dire que tout critère normalisé fournit la distance par rapport au meilleur concurrent.

$$z_i = -\frac{x_i - max(x)}{max(x) - min(x)} \tag{5.2}$$

Exemple La latence des services cloud représente souvent une part importante du temps global que les utilisateurs doivent attendre pour obtenir une réponse à leur requête. Or, plus la latence est importante, plus les performances sont faibles. Supposons que pour un stockage de données, la valeur de latence la plus faible pour une instance est de 6 ms et la plus élevée est de 287 ms. Dans ce cas, $(min(x)/max(x)) = (6/287) \geq 0.01$. Pour normaliser les valeurs de latence, nous utiliserons la formule 5.2.

47. un résultat que nous avons jugé raisonnable pour atteindre des valeurs normalisées non négligeables, tout en préservant l'importance de la différence entre les valeurs des NFA

48. un résultat que nous avons jugé raisonnable pour atteindre des valeurs normalisées non négligeables, tout en préservant l'importance de la différence entre les valeurs des NFA

Le résultat de la normalisation d'une valeur de latence de 20 ms est le suivant :

$$z_i = -\frac{20-287}{287-6} = 0,95$$

3. Dans le cas où les contraintes C1 et C4 sont respectées, pour tout critère normalisé, l'application de l'équation (5.3) fournit la distance par rapport au meilleur concurrent.

$$z_i = \frac{x_i}{\sqrt{\sum_{i=1}^{n} x_i^2}} \tag{5.3}$$

4. Dans le cas où les contraintes C2 et C4 sont respectées, nous appliquons l'équation (5.3) [139] et ensuite nous appliquons l'équation (5.4) pour obtenir la distance par rapport au pire concurrent.

$$z_i = 1 - \frac{n_i}{max(n)} \tag{5.4}$$

où n_i est la valeur normalisée de x_i dans le cas où la valeur maximale est meilleure pour le service (voir cas 3). n_i est calculé en utilisant l'équation (3). $max(n)$ est la valeur la plus élevée de l'ensemble n, avec n est l'ensemble des valeurs normalisées calculé pour l'ensemble x dans le cas 3.

Exemple Supposons que nous ayons un ensemble de 4 plans pour un service cloud donné. Le premier coûte 9000 dollars, le deuxième 10000 dollars, le troisième 5000 dollars et le quatrième 90 dollars. Les architectes veulent minimiser le prix de leurs services et maximiser les autres NFR tels que la performance, la sécurité et la disponibilité. Dans ce cas, $((min(x)/max(x)) = (90/10000) \leq 0.01)$.

Pour normaliser les valeurs des prix, nous utiliserons la formule (5.3) ensuite la formule (5.4).

Le résultat de la normalisation d'une valeur de prix de $x_i = 90$ dollars avec la formule (5.3) est le suivant :

$$\frac{90}{\sqrt{9000^2+10000^2+5000^2+90^2}} = 0.0062$$

Le résultat de la normalisation d'une valeur de prix de $max(x) = 10000$ dollars avec la formule (5.3) est le suivant :

$$\frac{10000}{\sqrt{9000^2+10000^2+5000^2+90^2}} = 0.70$$

À partir du résultat obtenu ci-dessus et en utilisant la formule (5.4), nous obtenons la valeur suivante pour la normalisation du prix de 90 dollars.

$$1 - \frac{0.0062}{0.70} = 0.91$$

Dans cette section, nous avons proposé des méthodes de normalisation pour normaliser les valeurs des NFA. Ces méthodes réduisent les différences entre les valeurs normalisées des NFA de sorte que ces valeurs normalisées reflètent la différence entre les valeurs des NFA sans les négliger.

L'étape suivante consiste à utiliser la méthode de prise de décision OWA pour agréger les valeurs des NFA et des NFR afin d'évaluer les services cloud en fonction des exigences des architectes. Elle utilise les valeurs normalisées des NFA recueillies lors de l'étape présentée dans cette sous-section.

5.3.3.2 Méthode de décision OWA pour la classification des services cloud

Une fois que nous avons normalisé les valeurs des NFA et recueilli leurs facteurs d'influence sur les NFR, le calcul d'une valeur globale pour chaque NFR est nécessaire pour évaluer les services en fonction des besoins des architectes. Nous sommes confrontés à un problème de prise de décision multicritères. Notre problème nécessite une méthode de prise de décision qui considère deux dimensions : d'abord, le poids d'importance des NFA et des NFR fournis par les architectes pour les services. Deuxièmement, la compensation entre les valeurs NFA et NFR les plus et les moins satisfaisants. Ainsi, nous éliminons les solutions qui peuvent avoir des valeurs très satisfaisantes pour certains NFA ou NFR et peu satisfaisantes pour d'autres. La seule méthode d'aide à la décision qui répond à cette exigence est l'opérateur OWA (Ordered Weighted Averaging) [140] qui fournit une fonction d'agrégation qui répond à ces exigences. Cet opérateur est défini comme suit :

Definition 5.3.1 *Soit p un vecteur de pondération de dimension n ($p = [\ p_1\ p_2\ ...\ p_n\ ]$) tel que : $p_i \in [0,1]$ et $\sum_i^n p_i = 1$. Une correspondance $OWA_p : R^n \longrightarrow R$ est un opérateur OWA de dimension n si*

$$OWA_p(a_1, a_2, ..., a_n) = \sum_i^n p_i a_\sigma(i) \tag{5.5}$$

où $\sigma(1), \sigma(2), ..., \sigma(n)$ est une permutation de $\{1, ..., n\}$ telle que $a_{\sigma(i-1)} \geq a_{\sigma(i)}$, pour

tous i = {2, ...,n} (c'est-à-dire que $a_\sigma(i)$ est le ième élément le plus grand de la collection a_1, ..., a_n).

Pour évaluer les services cloud, nous recourons à la méthode OWA selon les étapes suivantes :

1. l'application de l'algorithme OWA pour calculer les valeurs des NFA en fonction d'autres NFA dont l'importance a été déterminée par les architectes selon le type de leur application. Par exemple, la valeur du NFA "sauvegarde" est calculée en fonction de l'importance des NFA suivantes : fenêtre de sauvegarde, fréquence de sauvegarde et rétention de la sauvegarde.
2. une fois les valeurs NFA calculées, on applique la méthode OWA pour calculer les valeurs NFR. Les valeurs NFR sont calculées en agrégeant les valeurs NFA en fonction de leur influence (positive ou négative ou neutre) et de l'importance qui leur est attribuée par les architectes en fonction de l'application qu'ils veulent déployer.
3. l'application de l'algorithme OWA permet de calculer un score global pour chaque service. Ce score est calculé en agrégeant les valeurs NFR (sécurité, disponibilité, ...) selon l'importance que les architectes leur accordent en fonction de leurs types d'applications.

Le résultat de cette méthode est un score d'agrégation pour un NFA, un NFR ou un service. La méthode OWA sera utilisée comme opérateur d'agrégation dans l'algorithme que nous proposons dans la prochaine sous-section.

5.3.3.3 Algorithme de sélection des services

Nous proposons une approche de sélection des services pour rechercher un ensemble de plans de services adaptés aux besoins des architectes. Le coût de chacun des plans proposés ne doit pas dépasser un prix prédéfini par les architectes. Ces derniers peuvent ainsi choisir parmi l'ensemble des plans proposés par notre approche celui qui répond le mieux à leurs besoins. Les besoins des architectes en matière de services sont exprimés en termes d'influence des NFA sur les NFR et en termes d'influence des NFR sur le besoin du service (exprimés en poids d'importance). Notre approche comprend deux algorithmes, le premier consiste à construire l'arbre des NFA/NFR (algorithme 1) et le second à construire l'arbre pour chacun de ces NFA (algorithme 2). Dans cette section, nous présentons nos

Nom de variables	Type	Description
$\{NFA_i\}$	Liste	Contient les valeurs des NFA pour les plans de Service
P_s	Liste	Contient les plans de services
$\{NFR_j\}$	Liste	Contient les NFR à calculer
W_s	Vecteur	Contient les poids d'importance de chaque NFR
cost	Réel	Le prix à ne pas dépasser lors de la sélection des plans de Service
$\{[b_{min_i}, b_{max_i}]\}$	Liste	Contient les valeurs de bornes de normalisation pour les NFA
$\{w_{p_s}\}$	Liste	Contient les vecteurs de poids d'importance pour les NFA
evaluationPlansService	Structure arbre	Pointe vers une structure arbre qui contient : - valeurs score NFA après agrégation - valeurs score NFR après agrégation - valeurs score total des plans (triées par ordre décroissant) qui respecte la contrainte prix. - Nom et prix des plans du service.
listeNFAs	Liste des identifiants (NFA)	Liste des NFA racines pour un seul NFR.
listeNFAsNormalisée	Liste des identifiants (NFA)	Liste des valeurs NFA racine normalisée pour un seul NFR
liste_NFRs	Liste des identifiants (NFA)	Liste des NFA racines du service
noeudsNFARacine	Liste	Liste de noeud d'un NFA racine (Tête d'arbre). Chaque noeud contient les valeurs qui correspondent à une arborescence de poids.
noeudsNFR	Liste	Liste de noeud d'un NFR (Tête d'arbre). Chaque noeud contient les valeurs qui correspondent à une arborescence de poids.
Listecost	Liste	Liste qui contient les prix de tous les plans de service.

TABLE 5.2 – Spécification des variables de l'algorithme constructionArbres 1

algorithmes de sélection des services cloud.

Algorithme de sélection des services Les principales étapes de l'approche sont présentées dans l'algorithme 1. La description des variables et des fonctions de l'algorithme sont expliquées respectivement dans les tableaux 5.2 et 5.3.

Pour un service donnée s, l'algorithme prend en entrée, la liste des NFR, la liste des plans de services, la liste des NFA, le vecteur de poids d'importance W_s pour les NFR, les poids d'importance pour les NFA, les valeurs de bornes de normalisation pour les NFA et le coût à ne pas dépasser pour la sélection des plans de services. Il fournit en sortie une liste de plans de services ordonnées selon les scores calculés à partir des valeurs NFR en fonction de la contrainte de coût.

Notre algorithme comporte deux étapes. La première étape est la construction de l'arbre NFA/NFR. Dans cette étape, l'algorithme récupère d'abord les listes des NFA qui appartiennent au service (ligne 5). Ensuite, il récupère pour chaque NFR la liste des NFA racines qui appartiennent au NFR sélectionné (ligne 6 - ligne 8). Puis, il construit l'arbre pour chacun de ces NFA en utilisant **l'algorithme 2** qui attache les nœuds composites aux nœuds fils (l'algorithme est détaillé plus bas) (ligne 9 - ligne 12). Une fois l'arbre des

NFA construit jusqu'aux racines, l'algorithme normalise les valeurs des NFA racines (ligne 13). Ensuite, il calcule un score pour chaque NFR en utilisant les valeurs normalisées des NFA racines et leurs poids d'importance qui sont données comme input (ligne 14). Par la suite, l'algorithme construit l'arbre pour le service en attachant l'arbre des NFA au service correspondant (nœuds de service) (ligne 15).

La deuxième étape consiste à utiliser la méthode OWA pour calculer un score permettant de sélectionner les services les plus appropriés pour les architectes, de manière à ce que le prix ne dépasse pas un seuil. Dans cette étape, l'algorithme commence par lire les valeurs de prix des plans de services (données en INPUT) (ligne 17) et ensuite sélectionner les meilleurs services en fonction de la contrainte de coût (ligne 18).

Algorithm 1 Algorithme de sélection des services : constructionArbres

Inputs :
$\{\text{NFA}_i\}, P_s, \{NFR_j\}, W_s, cost, \{[\text{b}_{min_i}, b_{max_i}]\}, \{w_{p_s}\}$
Outputs :
evaluationPlansService
BEGIN
1: listeNFAs ← null ;
2: listeNFAsNormalisée ← null ;
3: evaluationPlansService ← null ;
4: noeudsNFARacine ← null ;
5: liste_NFRs ← *listeNFRs*($\{\text{NFR}_j\}$)
6: **for each** NFR in liste_NFRs **do**
7: noeudsNFR ← NFR ;
8: listeNFAs ← *listeNFAs*(NFR,P_s,$\{NFA_i\}$) ;
9: **for each** NFA in listeNFAs **do**
10: noeudsNFARacine ← **constructionArbreNFA**(NFA) ;
11: noeudsNFR.*add*(noeudsNFARacine) ;
12: **end for each**
13: listeNFAsNormalisée ← *NormaliserNFAsRacine*(listeNFAs,$\{[b_{min_i}, b_{max_i}]\}$) ;
14: noeudsNFR ← *OWA*(listeNFAsNormalisée, $\{w_{p_s}\}$) ;
15: evaluationPlansService.*add*(noeudsNFR) ;
16: **end for each**
17: listecost ← lirePrix($\{P_s\}$) ;
18: evaluationPlansService ← *Select* (*OWA*(liste_NFRs, W_s, *cost),listecost)* ;
END

Algorithme de construction d'arbre de NFA L'algorithme 2 présente l'approche pour construire l'arbre des NFA. La description des variables et des fonctions de l'algorithme sont expliquées respectivement dans les tableaux 5.5 et 5.4. Cet algorithme prend en entrée un NFA. L'algorithme récupère d'abord les listes des sous-NFA (fils) du NFA donné en input (ligne 4), si celui-ci est un NFA composite. Dans le cas contraire, il renvoie

Nom fonction	Entrées	Sorties	Description
listeNFRs	$\{NFR_j\}$	Liste des identifiants (NFR)	Récupérer la liste des identifiants NFR du service spécifié en entrée
listeNFAs	Identifiant NFR, P_s, $\{NFA_i\}$	Liste des identifiants NFA	Récupérer la liste des identifiants NFA qui compose le NFR spécifié en entrée.
constructionArbreNFA	Voir algorithme plus bas 'constructionArbreNFA'		
add	1. Noeud père (service/NFR/NFA) 2. Noeud fils (NFA/NFA)	Noeud père (service/NFR/NFA)	Attache le noeud fils
NormaliserNFAsRacine	Listes des valeurs de NFA racines	Liste des valeurs NFA racine normalisée	Normalisation des valeurs des NFA racines
OWA	Liste de valeurs normalisée + poids d'importance	Liste de valeur de score agrégé des paramètres en entrée pour les plans de services	Agrégation de valeurs
Select	Scores NFR obtenus avec OWA + contraintes sur prix	Liste de valeurs de score des meilleurs plans de service	Obtenir les meilleurs plans de service par rapport à un prix seuil

TABLE 5.3 – Spécification des fonctions de l'algorithme constructionArbres 1

cet NFA directement au premier algorithme. Ensuite, pour chaque sous-NFA, il effectue un appel récursif pour renvoyer le nœud (fils) du NFA donné (ligne 5 - ligne 7). Ce processus est répété jusqu'à ce qu'il atteigne les feuilles de l'arbre (les NFA non composites). Ensuite, il construit l'arbre en attachant chacun des NFA composites à ses sous-NFA (fils) (ligne 8). Une fois que l'arbre des NFA est construit depuis les nœuds des feuilles jusqu'aux nœuds des racines, l'algorithme normalise les valeurs de tous les NFA (composites et non composites) (ligne 14). Enfin, la méthode OWA est utilisée pour le calcul d'un score de chaque NFA composite (ligne 15). Le résultat de cet algorithme est un arbre de NFA avec leurs valeurs agrégées.

Algorithm 2 Algorithme de construction d'arbre NFA : constructionArbreNFA

Inputs :
NFAInput
Outputs :
ArbreNFA : arbre
BEGIN

```
1: ArbreNFAs ← NFAInput ;
2: ArbreNFANormaliser ← null ;
3: listeNFAsCorrespondant ← getListeNFAsCorrespondant(NFAInput) ;
4: if (listeNFAsCorrespondant != null (si il a des fils ) then
5:    for each NFA in listeNFAsCorrespondant  do
6:       [ArbreSousNFA ← constructionArbreNFA(NFA) ;
7:       ArbreNFAs ← attacherArbre(ArbreNFAs , ArbreSousNFA) ;
8:    end for each
9:    ArbreNFANormaliser ← Normaliser(ArbreNFAs) ;
10:   ArbreNFAs ← OWA(ArbreNFANormaliser, poids(ArbreNFAs)) ;
11: end if
```

END

NOM Variables	TYPE	Description
NFAInput	Identifiant d'un NFA	Un NFA Input
ArbreNFAs	Arbre	Arbre d'un NFA Input
listeNFAsCorrespondant	Liste des identifiants (NFA)	Liste des sous NFA pour un NFA (récursive)
ArbreSousNFA	Arbre	Arbre d'un sous NFA
ArbreNFANormalise	Arbre	Liste des valeurs NFA (et sous NFA) normalisée

TABLE 5.5 – Spécification des variables de l'algorithme 2

Noms des fonctions	Entrées	Sorties	Description
getListeNFAsCorrespondant	Identifiant NFA	Liste NFA Fils (sous NFA)	Renvoie une liste qui contient les sous NFA (les NFA fils) du NFA passés en paramètre (si ce dernier n'a pas de sous NFA, la liste renvoyée sera vide)
attacherArbre	Nœud père (NFA) Nœud fils (NFA/NFA)	Nœud père (NFA)	Attache l'arbre (sous NFA / NFA)
Normaliser	Listes des valeurs de NFA	Liste des valeurs NFA normalisée	Normalisation des valeurs des NFA
poids	Liste des identifiants (NFA)	poids d'importance	Récupérer les poids d'importances des NFA
OWA	Liste de valeurs normalisée des NFA + poids d'importance de ses NFA	Liste de valeur de score agrégé des paramètres en entrée pour les plans de services.	Agrégation de valeurs

TABLE 5.4 – Spécification des fonctions de l'algorithme 2

Dans cette section, nous avons proposé une approche de la sélection des services cloud. Cette approche consiste à utiliser des NFA réels du marché collectés auprès des fournisseurs de services cloud et de leur documentation, ainsi que des benchmarks et des avis d'archi-

tectes cloud. Nous pouvons séparer notre étude en deux parties, la première consiste à déterminer l'impact des NFA sur les NFR en fonction des types de services. La deuxième partie consiste à utiliser les méthodes de décision appropriées et à proposer des algorithmes pour déterminer les plans de service les mieux adaptés aux besoins des architectes.

La détermination de l'impact des NFA sur les NFR est liée, dans un premier temps, à la mise en place d'une enquête auprès des architectes afin d'attribuer des poids significatifs aux NFA/NFR des services. Dans un deuxième temps, comme la performance est un NFR particulier, nous avons proposé de déterminer l'impact des NFA sur la performance en cherchant à partir des études de benchmarks et de référence des indicateurs permettant d'évaluer la performance des services cloud.

Quant à la seconde partie, elle s'intéresse à la manière de traiter les données (NFA/NFR) collectées dans la première partie afin de sélectionner les meilleurs plans de service selon les besoins et le prix spécifiés par les architectes. Ainsi, nous avons utilisé des méthodes de normalisation appropriées pour traiter les données d'une manière pertinente pour notre problème. Ensuite, nous avons proposé un algorithme de sélection de services qui incorpore la méthode de décision OWA. Le choix de cette méthode est dû au fait qu'elle est, à notre connaissance, la seule dans la littérature qui permet d'éliminer des solutions qui peuvent avoir des valeurs très satisfaisantes pour certains NFA ou NFR et des valeurs insatisfaisantes pour d'autres. Nous présentons dans la prochaine section la validation de notre méthode sur différents cas d'utilisation.

5.4 Validation de la méthode de sélection de service

Nous validons la pertinence de notre approche de sélection des services cloud pour deux cas d'utilisation à savoir le service de base de données relationnelle et le service file de messagerie pour différentes capacités et différents besoins de service. Nous évaluons notre approche par rapport à un plan de service sélectionné par les architectes parmi les k meilleurs plans proposés par notre approche.

5.4.1 Étude de cas 1 : services cloud de bases de données relationnelles

Pour valider notre approche, nous l'avons appliquée à la sélection des services cloud de base de données SQL. Dans nos expérimentations, nous avons considéré les services des

fournisseurs Amazon et Azure situés en Europe, avec une capacité de stockage de 64 GB par mois. Ces services doivent répondre aux besoins de chacun des micro-services décrits dans l'exemple de motivation 1.1. Nous avons sélectionné les plans de service les plus appropriés en fonction des exigences des architectes en termes de NFR et de prix.

5.4.1.1 Résultats de l'enquête pour attribuer des poids significatifs aux NFA et aux NFR en fonction des besoins des architectes en matière d'applications

Nous avons mené une étude empirique avec les architectes sur les services cloud de base de données relationnelles, comme mentionné dans la sous-section 5.3.1. Nous avons utilisé cette méthodologie pour interroger les architectes et déterminer l'importance des NFA et des NFR. Les architectes ont toujours répondu : "selon le contexte du projet".

Par exemple, nous avons demandé aux architectes d'exprimer leurs besoins en termes de backups pour le service SQL de leur projet. Tous les architectes ont exprimé leurs exigences de la même manière. Ils s'attendaient à ce que les sauvegardes soient conservées le plus longtemps possible et à ce que les backups soient effectués quotidiennement. Les architectes attribuent des notes allant de 10 à 20 points pour les backups manuels, de 40 à 60 points pour la conservation des backups, de 15 à 30 points pour la fenêtre de backup et de 10 à 20 points pour le nombre de backups par jour. Nous avons posé la même question pour le service SQL d'une application qui traite les commentaires des utilisateurs en décryptant leurs opinions. Les architectes ont prévu une période de conservation des données de backups de 7 jours et une fréquence de backups de deux fois par jour. Ils attribuent des notes allant de 5 à 10 points pour les sauvegardes manuelles, de 20 à 40 points pour la conservation des sauvegardes, de 15 à 25 points pour la fenêtre de backups, et de 20 à 30 points pour le nombre de backups par jour. Nous avons remarqué que dans toutes les questions que nous avons abordées avec les architectes au sujet des NFA et de leurs influences sur les NFR, ils ont systématiquement mentionné que *"les exigences des services dépendent de l'application que nous voulons déployer"*. À chaque fois, ils ont donné des exemples d'exigences différentes en fonction du contexte d'utilisation des services. Ces entretiens nous ont permis de conclure que les besoins des architectes en matière de services dépendent du type d'application qu'ils veulent déployer. Par conséquent, nous pensons que la méthodologie d'évaluation des services peut être développée en utilisant des patterns

propres à la nature des applications. Ces patterns peuvent être fournis par les architectes.

La prochaine sous-section présente une évaluation de la mesure de l'impact des NFA sur la performance. Une attention particulière doit être accordée à la performance, c'est le NFR le plus difficile à évaluer.

5.4.1.2 Résultats liées à l'incapacité de mesurer l'impact des NFA sur les performances

Les fournisseurs de services Amazon et Azure fournissent des informations sur les configurations de déploiement de leurs services, telles que le nombre de processeurs utilisés pour leurs instances. Cependant, l'influence du nombre de processeurs sur les performances n'est pas connue. En effet, cette influence dépend des caractéristiques des processeurs telles que leur type (Intel Xeon Platinum, AMD,...) et leur puissance (exprimée en GHz). Afin de permettre aux développeurs de comparer facilement la puissance de calcul des processeurs entre différents types d'instances, Amazon fournit des unités de calcul élastiques (ECU : Elastic Compute Units). Nous avons évalué les CPU des instances sur la base de cette unité de calcul (ECU). Nous avons considéré le fait que les fournisseurs de services fournissent une augmentation linéaire de la capacité ECUs des instances en fonction du prix (c'est-à-dire que plus vous payez, plus vous avez de capacité ECUs). Nous avons adopté la démarche suivante : (1) d'abord, nous recueillons les informations sur les ECUs disponibles sur la plateforme Amazon. (2) ensuite, nous comparons chaque instance Amazon dont la capacité ECU est connue avec les instances Amazon dont la capacité ECU n'est pas connue. De cette façon, le nombre des ECUs pour tous les plans d'Amazon sont déterminés. (3) enfin, comme le nombre des ECUs est une capacité spécifiée uniquement pour Amazon, nous devons convertir la capacité des CPUs d'autres fournisseurs en ECUs. Pour cela, nous avons examiné des benchmarks [137] afin de comparer les performances des instances d'autres fournisseurs avec celles d'Amazon. Ainsi, nous avons déterminé le nombre des ECUs à attribuer aux instances d'autres fournisseurs. Nous avons constaté également dans cette étude que la performance des services de cloud relationnel dépend fortement du processeur et légèrement du nombre d'IOPS, de SSD et d'autres paramètres.

Dans les prochaines sous-sections, nous présentons l'expérimentation et la validation de notre approche.

Micro-services /NFR	SQL du micro-service 1	SQL du micro-service 2	SQL du micro-service 3 (1)	SQL SQL du micro-service 3 (2)
Disponibilité	20 points	25 points	10 points	20 points
Fiabilité	20 points	10 points	10 points	10 points
Performance	5 points	27 points	40 points	30 points
Scalabilité de la capacité	5 points	5 points	10 points	10 points
Sécurité	30 points	23 points	10 points	10 points
Coût	20 points	10 points	20 points	20 points
Poids total des colonnes	100 points	100 points	100 points	100 points

TABLE 5.6 – Exigences du service SQL pour chaque micro-service

5.4.1.3 Expérimentation et résultats

Nous avons examiné avec les architectes les poids d'importance des NFA et des NFR en fonction de leurs exigences pour les services SQL des 3 micro-services décrits dans la section 1.1. Les architectes ont 100 points à distribuer entre les NFR et entre les NFA pour les différents services SQL. Ces points sont attribués en fonction des exigences de chacun des micro-services. Le tableau 5.6 illustre les résultats de cette étude. Pour les Micro-Services 1 et 2, une seule exigence a été interprétée et pour le Micro-Service 3, deux exigences ont été considérées.

En appliquant notre méthodologie à partir des besoins des architectes exprimés dans le tableau 5.6, nous avons classé les plans de services SQL pour chaque micro-service. La figure 5.3 montre les résultats des meilleurs plans de service SQL obtenus pour chaque micro service.

5.4.1.4 Discussion

La validation de la pertinence de la sélection des services cloud est une tâche difficile. Premièrement, pour certains NFR, la validation n'est pas toujours possible. Par exemple, les prestataires de services mettent en œuvre des mesures pour sécuriser leurs services. Ces mesures n'ont pas empêché certains services de rencontrer des problèmes de sécurité causés par une vulnérabilité que les concepteurs du service ne pouvaient pas anticiper. Ensuite, il y a les NFR pour lesquels la validation n'est valable que pour des scénarios spécifiques et sous des contraintes spécifiques. De plus, la mise en œuvre de tests pour ces NFR est très coûteuse et prend beaucoup de temps. Pour ces raisons, nous évaluons les NFR des services sur la base des informations disponibles auprès des prestataires de services et les validons avec les architectes.

NFA	IOPS	SSD	ECU	RAM	tenancy	Backups	Deployment	Read Replicas
Poids	0.035	0.015	0.475	0.4	0.025	0.025	0.125	0.125
Impact	+	+	+	+	- :Multi tenancy. + :Single tenancy.	- :Exists = :Doesn't exist	- :MultiAZ = :MonoAZ	+ : Synchronous = : Semisynchronous - : Asynchronous

TABLE 5.7 – Résultats de l'évaluation des attributs de performance et leurs impacts

Attributs	Deployment	Read Replicas
Significance	0.85	0.15
Impact	+ : MultiAZ = : MonoAZ	+ : Exists = : Doesn't exist

TABLE 5.8 – Résultats de l'évaluation des attributs de disponibilité et leurs impacts

Nous visons à valider notre approche par rapport aux exigences de service présentées dans le tableau 5.6. Pour ce faire, nous utilisons les configurations des plans de service. Les plans sont similaires pour tous les attributs d'un même fournisseur de services (Amazon ou Azure), sauf en ce qui concerne les performances et la disponibilité. Tout d'abord, nous avons recueilli les influences des NFA sur les NFR à partir de l'étude menée dans le chapitre 4. Ensuite, nous avons déterminé l'importance des NFA sur la performance en utilisant les benchmarks [113] [10]. Nous avons également déterminé l'importance des NFA sur la disponibilité en interrogeant des architectes qui nous ont donnés une indication sur leur impact. Les résultats des valeurs d'importance attribuées aux NFA et leur impact sur la performance et la disponibilité sont présentés dans les tableaux 5.7 et 5.8 respectivement. Par exemple, le type de déploiement et les reads replicas affectent la disponibilité. Le déploiement multi-AZ et la présence des reads replicas améliorent la disponibilité (cette influence est représentée dans le tableau 5.8 par un +). Le coefficient d'influence des reads replicas sur la disponibilité est de 0,15 et celui du déploiement est de 0,85.

Nous avons sélectionné les plans de service SQL pertinents en fonction des exigences de chaque micro-service. Les résultats sont validés par les architectes sur la base de la configuration des plans, qui sont présentés dans le tableau 5.9. Le micro-service 1 nécessite des performances et une scalabilité faibles, une très haute sécurité (chiffrement au repos, VPC et single tenant), une haute disponibilité (multiAZ + Read Replicas) et une haute fiabilité pour un prix réduit. Comme le montre la figure 5.3, ce plan correspond aux exigences du micro-service 1 pour seulement $39 par mois. Le micro-service 2 nécessite une haute disponibilité, une haute sécurité, des performances élevées, une faible fiabilité, une faible scalabilité et un coût acceptable. Comme le montre la figure 5.3, ce plan correspond aux

exigences du micro-service 2, une grande disponibilité en multiAZ avec des répliques de lecture et un niveau de performance relativement bon selon le nombre d'ECUs, la quantité de RAM et le nombre d'IOPS pour un coût de $305 par mois. Le micro-service 3 pour l'exigence 1 exige des performances élevées ainsi qu'une sécurité, une fiabilité, une scalabilité et une disponibilité réduites à un prix abordable. Comme le montre la figure 5.3, ce plan correspond parfaitement à l'exigence 1 du micro-service 3 qui ne nécessite qu'une réplication monoAZ avec des répliques en lecture et des paramètres de haute performance : 244 Go de RAM, 140 unités de calcul ECU et 1000 IOPS pour un coût de $3167 par mois. Le micro-service 3 pour l'exigence 2 nécessite des performances élevées, une sécurité réduite, une fiabilité réduite, une scalabilité réduite et une haute disponibilité pour un prix raisonnable. Comme le montre la figure 5.3, cela correspond aux besoins du micro-service 3 pour l'exigence 2, une haute disponibilité en multiAZ avec des répliques en lecture et des paramètres de performance satisfaisants : 15 Go de RAM, 1 unité de calcul ECU et 1000 IOPS pour un coût de $305 par mois, soit la même configuration que pour le micro-service 2. Pour cet exemple, nous voyons que la méthode que nous proposons peut aider l'architecte à sélectionner des configurations adaptées avec des critères discriminants. Cette méthode semble prometteuse bien qu'elle nécessite une validation supplémentaire.

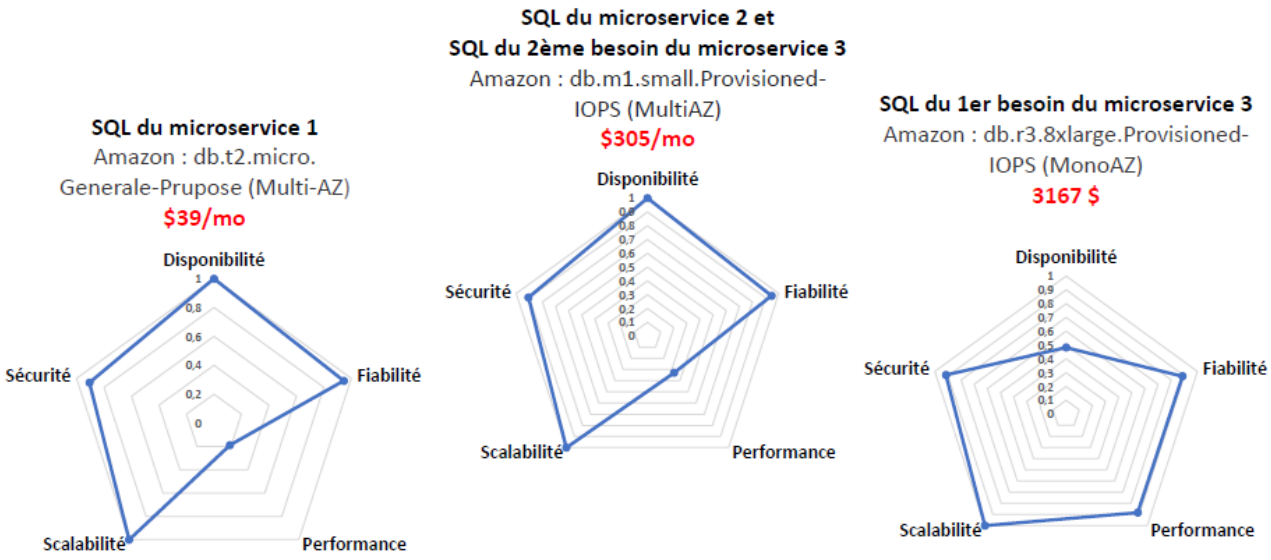

FIGURE 5.3 – Sélection de plans SQL pour les micro-services

Dans cette sous-section, nous avons sélectionné les meilleurs services SQL pour une exigence de 64 Go, qui répondent aux exigences spécifiées par les architectes dans le tableau

SQL Services	db.t2.micro	db.m1.small	db.r3.8xlarge
Instance type	General Purpose	Provisioned IOPS	Provisioned IOPS
deployment	Multi AZ	Multi AZ	Mono AZ
Read replicas	exist	exist	exist
Replication Type	Semi synchronous	Semi synchronous	Semi synchronous
Tenants	Single-tenant	Single-tenant	Single-tenant
Backup	exist	exist	exist
IOPS	192	1000	1000
SSD	exist	exist	exist
RAM	1GB	15GB	244GB
ECUs	0.03125	1	140
Scaling option	to another storage type	to another storage type	to another storage type
VPC	exist	exist	exist
Encryption At Rest	exist	exist	exist
Price/month	39$	305$	3167$

TABLE 5.9 – Service SQL - configuration des plans

5.6. Nous avons validé le choix approprié des services en collaboration avec les architectes sur la base de la configuration des plans. Ces expérimentations démontrent les défis à relever pour sélectionner efficacement les services pour l'architecture des micro-services. Tout d'abord, il est difficile de comparer les plans de services proposés par les fournisseurs en fonction de leurs performances et de leur déploiement. D'autres attributs de configuration tels que la sécurité (VPC et chiffrement au repos) et la fiabilité (sauvegarde de rétention, nombre de sauvegardes) sont similaires quel que soit le plan proposé par ces fournisseurs (Amazon et Azure). Deuxièmement, les plans des différents fournisseurs de services (Amazon et Azure) varient grandement en fonction des valeurs de configuration de la scalabilité (le stockage personnalisé peut être mis à l'échelle ou converti en un autre type de stockage pour Amazon et n'est pas pris en charge par Azure), de la fiabilité (conservation des backups, nombre de backups)), et ainsi de suite. La troisième observation est que le coût des plans Amazon varie considérablement en fonction du déploiement (multi-AZ ou mono-AZ) et des performances, tandis que pour Azure, le coût des plans varie fortement en fonction des performances et légèrement en fonction du déploiement. Néanmoins, il est possible de suivre notre méthode pour obtenir des conseils dans la sélection de plans spécifiques basés sur une évaluation des critères NFR. Il faudrait également former les architectes pour les aider à considérer plus systématiquement l'importance des NFR pour leurs architectures.

5.4.2 Étude de cas 2 : service cloud de file d'attente

Pour valider notre approche, nous l'avons appliqué à la sélection des services cloud de file d'attente. Dans nos expérimentations, nous avons considéré les services localisés en Europe des fournisseurs suivants : Heroku- IronMQ , Heroku-CloudAMQP, cloudamqp et Amazon SQS. Nous avons étudié les files d'attente de type FIFO avec une capacité minimale de 100 messages par file d'attente, un nombre de files d'attente de 1000, un nombre de connexions de 100 et un nombre de requêtes de 10.000.000 par mois.

Nous avons sélectionné les plans de service les plus appropriés en fonction des exigences des architectes en termes de NFR et de prix.

5.4.2.1 Résultats de l'enquête pour attribuer des poids significatifs aux NFA et aux NFR en fonction des besoins des architectes en matière d'applications

Nous avons mené une étude empirique avec les architectes sur les services cloud de file d'attente, comme mentionné dans la sous-section 5.3.1. Nous avons utilisé cette méthodologie pour interroger les architectes et déterminer l'importance des NFA et des NFR. Les architectes ont toujours répondu : "selon le contexte du projet". Par exemple, nous avons demandé aux architectes d'exprimer leurs besoins en termes de fiabilité pour le service de file d'attente de leur projet. Tous les architectes ont exprimé leurs exigences de la même manière. Ils s'attendaient à ce que les messages soient persistants pour éviter de perdre ces messages en cas de redémarrage du broker, de panne de matériel ou de crash du broker. Ils exigent également que les files d'attente de messages et tous les messages soient hautement disponibles, comprenant plusieurs zones de disponibilité (AZ) redondantes afin qu'aucune défaillance de réseau ou de zone de disponibilité ne puisse empêcher l'accès aux messages. En outre, ils précisent que leur application exige que tous les messages transmis soient traités par les services concernés. Ainsi, des accusés de réception (ACK) sont nécessaires en cas de perte des messages en transit ou de défaillance de la connexion en vue de leur retransmission. Pour les partitions de réseau, les architectures préfèrent le mode "ignorer" pour leur application. Ce mode doit être utilisé lorsque la fiabilité du réseau est maximale et que la disponibilité des nœuds est de la plus haute importance. Ensuite, les architectures préfèrent le mode "pause_minorité" qui est approprié lorsque les clusters sont distribués sur des racks ou des zones de disponibilité dans une seule région, et que la probabilité de

perdre une majorité de nœuds (zones) en même temps est considérée comme très faible. Enfin, les architectes ne veulent pas du mode "autoheal" qui est approprié lorsqu'il y a plus de préoccupations concernant la continuité du service que la cohérence des données entre les nœuds. Les architectes attribuent des notes allant de 10 à 20 points pour les ACKs, 10 à 20 points pour le déploiement, 15 à 25 points pour le type tenant, 40 à 50 points pour la persistance et 10 à 20 points pour le partitionnement réseau. Nous avons posé la même question pour le service de file d'attente d'une application qui traite les commentaires des utilisateurs en décryptant leurs opinions. Les architectes prévoient une persistance des messages, un déploiement multi-AZ et n'exige pas d'ACK pour cette application. Ils attribuent des notes de 0 points pour les ACKs, de 20 à 40 points pour le déploiement multi-AZ, de 5 à 10 points pour le type de tenant, de 50 à 60 points pour la persistance des messages et de 10 à 20 pour le partitionnement réseau.

Les conclusions de cette étude sont les mêmes que celles de la section 5.4.1.

La prochaine sous-section présente une évaluation de la mesure de l'impact des NFA sur la performance. Une attention particulière doit être accordée à la performance, c'est le NFR le plus difficile à évaluer.

5.4.2.2 Résultats liés à l'incapacité de mesurer l'impact des NFA sur les performances

La plupart des fournisseurs de services cloud de file d'attente proposent le débit et le temps de réponse comme indicateurs de performance pour leurs services. Ces deux indicateurs de performance sont suffisants pour évaluer la performance des services cloud de file d'attente. Par conséquent, nous avons étudié les travaux de benchmarks [141] qui étudient le nombre de messages par seconde qu'un système de mise en file d'attente donné peut traiter la latence de traitement, qui est un facteur important dans les systèmes qui doivent réagir aux nouvelles données en temps quasi réel (comme c'est souvent le cas avec les différents flux d'événements). Un autre aspect important est la latence d'envoi des messages, c'est-à-dire le temps qu'il faut à un client pour être sûr qu'un message est persistant dans le système de mise en file d'attente. Cela peut avoir un impact direct, par exemple sur la latence des terminaux http et sur l'expérience de l'utilisateur final.

Les auteurs du [141] ont utilisé trois mesures pendant les tests :

— Le degré de messages par seconde : la vitesse moyenne de la file d'attente, c'est-à-

dire le nombre de messages par seconde qui peuvent être envoyés, et le nombre de messages par seconde qui peuvent être reçus et faire l'objet d'un accusé de réception

— latence de traitement du 95e percentile (sur une fenêtre d'une minute) : temps (en millisecondes) qui s'écoule entre l'envoi et la réception d'un message, c'est-à-dire la vitesse à laquelle le broker fait parvenir le message de l'expéditeur au destinataire.

— latence d'envoi (95e percentile) (dans une fenêtre d'une minute) : temps nécessaire pour envoyer un message. C'est le temps pendant lequel il est sûr que le message est persistant dans le groupe, et que celui-ci peut, par exemple, répondre à la requête http du client "message reçu".

Lors de la mise en place des files d'attente, il est prévu que trois nœuds identiques et répliqués fassent fonctionner le serveur des files d'attente de messages, avec un basculement automatique.

Chaque test est défini par le type de file d'attente des messages à tester, les paramètres optionnels de la file d'attente des messages, le nombre de nœuds clients, le nombre de threads sur chaque nœud client et le nombre de messages. Un nœud client envoie ou reçoit des messages. Dans les tests, les nœuds clients de chaque type sont au nombre de 1 à 8 (il y a toujours le même nombre de nœuds clients et de nœuds récepteurs), chacun d'entre eux ayant de 1 à 25 threads.

Chaque file d'envoi essaye d'envoyer le nombre de messages donné le plus rapidement possible, par lots de taille aléatoire entre 1 et 10 messages. Pour certaines files d'attente, la procédure d'évaluation de lots plus importants, jusqu'à 100 ou 1000 messages, a été mise en place

Le destinataire essaye de recevoir des messages (également par lots de 10 messages maximum), et après les avoir reçus, il en accuse réception (ce qui devrait entraîner le retrait du message de la file d'attente). Le test se termine lorsqu'aucun message n'est reçu pendant une minute.

Nous utilisons pour notre évaluation les valeurs moyennes de latence et de débit pour chaque fournisseur.

Par exemple, les temps de latence des fournisseurs de services [141] pour les différentes technologies de mise en file d'attente sont les suivants :

— RabbitMQ : 66ms

— SQS : 210ms

— ActiveMQ : 63ms

— Mongo : 48ms

— Artemis : 50ms

— Event Store : 66ms

— Iron MQ : 210ms

Dans les prochaines sous-sections, nous présentons l'expérimentation et la validation de notre approche.

5.4.2.3 Expérimentation et résultats

Nous avons examiné avec les architectes l'importance des NFA et des NFR dans le contexte de leurs exigences architecturales pour les services cloud de file d'attente. Les architectes ont 100 points à distribuer entre les NFR et entre les NFA pour les différents services de file d'attente. Le tableau 5.10 illustre les résultats de cette étude.

micro-services/NFR	Besoin 1 pour la file d'attente	Besoin 2 pour la file d'attente
Disponibilité	15 points	30 points
Fiabilité	15 points	40 points
Performance	25 points	10 points
Scalabilité de la capacité	20 points	10 points
Sécurité	25 points	10 points
Prix	le prix ne dépasse pas 4000 dollars	le prix ne dépasse pas 4000 dollars
Poids total des colognes	100 points	100 points

TABLE 5.10 – les exigences des services de file d'attente

En appliquant notre méthodologie basée sur les besoins des architectes, nous avons classé les plans de services de files d'attente pour répondre à chacun des besoins de l'application. La figure 5.4 montre les résultats des meilleurs plans de service file d'attente obtenus pour chaque besoin applicatif. Les architectes ont choisi cette solution principalement en raison des faibles coûts des services. Une autre solution alternative pour les besoins 1 et 2 des services de file d'attente est illustrée dans la figure 5.5

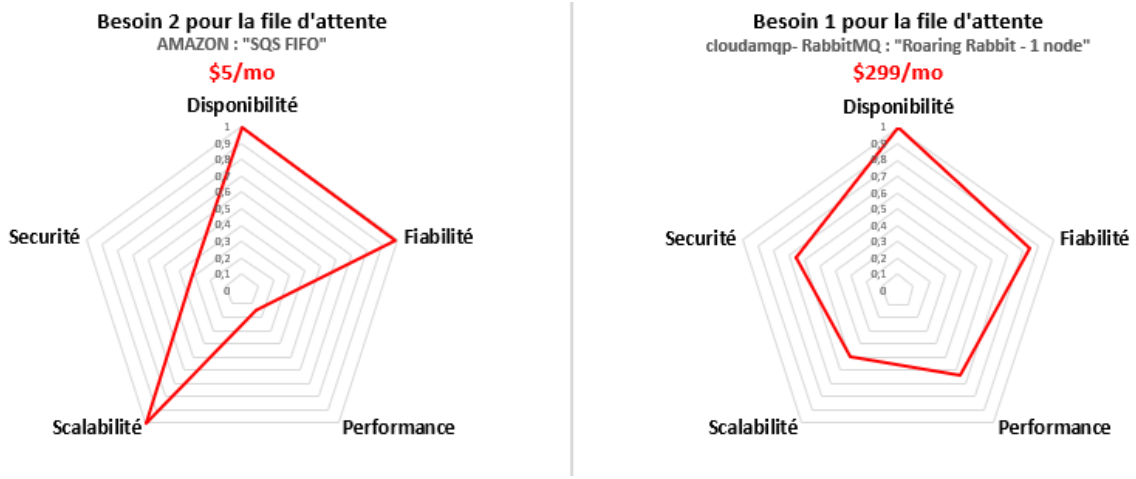

FIGURE 5.4 – Sélection de plans de file d'attente pour les besoins des architectes

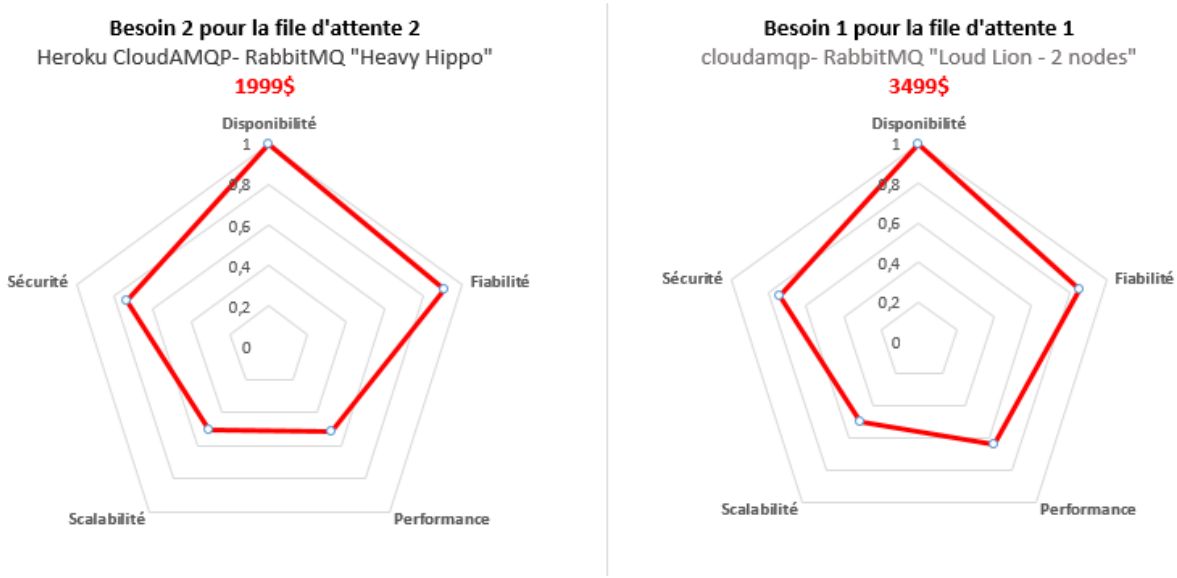

FIGURE 5.5 – Une alternative de sélection de plans de file d'attente pour les besoins des architectes

5.4.2.4 Discussion

Comme nous l'avons mentionné à la sous-section 5.4.1.4, la validation de la pertinence de la sélection des services cloud est une tâche difficile. Par conséquent, nous validons notre approche sur la base des informations disponibles auprès des fournisseurs de services et des opinions des architectes.

Tout d'abord, nous avons recueilli les influences des NFA sur les NFR à partir de l'étude menée dans le chapitre 4. Ensuite, nous avons déterminé l'importance des NFA sur la performance en utilisant les benchmarks [10] Nous avons également déterminé l'influence

des autres NFA sur les NFR en interrogeant des architectes qui nous ont donnés une indication sur leur impact. Les résultats des valeurs d'importance attribuées aux NFA et leur impact sur la sécurité, la fiabilité, la performance, la scalabilité et la disponibilité sont présentés dans les tableaux 5.11 ; 5.12 ; 5.13 ; 5.14 ; 5.15 respectivement.

Le tableau 5.11 montre que :

— l'existence du VPC pour un plan influence positivement la sécurité avec un degré de 0,5 sur 1.

— l'existence du chiffrement pour un plan influence positivement la sécurité avec un degré de 0,1 sur 1.

— l'existence de la persistance et du déploiement multi-AZ pour un plan a un impact négatif sur la sécurité d'un degré de 0,1 sur 1 chacun.

NFA	VPC	Encryption	Persistance	Deployment
Poids	0.5	0.3	0.1	0.1
Impact	+ : Existe = : N'existe pas	+ : Existe = : N'existe pas	- : Existe = : N'existe pas	- : MultiAZ = : MonoAZ

TABLE 5.11 – Résultats de l'évaluation des NFA en termes de sécurité et de leurs impacts

Le tableau 5.12 montre que :

— l'existence de ACKs pour un plan influence positivement la fiabilité avec un degré de 0,2 sur 1.

— l'existence du déploiement multi-AZ pour un plan influence positivement la fiabilité avec un degré de 0,2 sur 1.

— Les instances single-tenant pour un plan influence négativement la fiabilité avec un degré de 0,1 sur 1.

— Le partitionnement du réseau pour un plan aura une influence positive sur la fiabilité si le mode "ignore" est utilisé, négative sur la fiabilité si le mode "Autoheal" est utilisé, neutre si le mode "Pause Minority" est utilisé. Le degré d'influence du partitionnement du réseau sur la fiabilité est de 0,1 sur 1

— La persistance des messages a une influence positive sur la fiabilité. Sa valeur est calculée à partir de la rétention de la persistance des messages et du nombre de messages persistants. Le degré d'influence de la persistance sur la fiabilité est de 0,5 sur 1.

NFA	Ack	Deployment	Tenants	Persistance	Partitions Réseau
Poids	0.2	0.2	0.1	0.5	0.1
Impact	+ : Existe = : N'existe pas	+ : MultiAZ = : MonoAZ	- : Single-tenant = : Multi-tenant	Calculer à partir de la rétention de persistance des messages et le nombre de messages persistants	+ : Mode ignore = : Mode Pause Minority - : Mode Autoheal

TABLE 5.12 – Résultats de l'évaluation des NFA en termes de fiabilité et de leurs impacts

Le tableau 5.13 montre que le débit et le temps de réponse influencent les performances du plan. Un débit élevé et un temps de réponse faible augmentent les performances. L'influence du débit et du temps de réponse est de 0,7 et 0,3 sur 1, respectivement.

NFA	Débit (msg/s)	Latence
Poids	0.7	0.3
impact	+	-

TABLE 5.13 – Résultats de l'évaluation des NFA en termes de performance et de leurs impacts

Le tableau 5.14 montre que l'existence de la scalabilité verticale [49] et de la scalabilité horizontale [50] ont une influence positive sur la scalabilité avec un degré de 0,5 sur 1 chacun.

NFA	Scalabilité Verticale	Scalabilité Horizontale
Poids	0.5	0.5
Impact	+ : existe = : n'existe pas	+ : existe = : n'existe pas

TABLE 5.14 – Résultats de l'évaluation des NFA en termes de scalabilité et de leurs impacts

Le tableau 5.15 montre que le partitionnement du réseau [51] pour un plan aura une influence positive sur la disponibilité si le mode "Pause minority" est utilisé, négative si le mode "Mode ignore" est utilisé, neutre si le mode "Autoheal" est utilisé. Le degré d'influence du partitionnement du réseau sur la disponibilité est de 0,2 sur 1 et l'existence du déploiement multi-AZ pour un plan influence positivement la disponibilité avec un degré de 0,8 sur 1.

Comme les plans de services que nous avons collectés ne sont pas très efficaces sur certains NFR en raison de leur configuration (conditions de déploiement), nous avons donc

49. Scalabilité verticale ("scale-in") : consiste à augmenter les ressources des serveurs (processeur, mémoire...). Cette méthode est très efficace dans le cas de services qui peuvent se paralléliser.

50. Scalabilité horizontale ("scale-out") : consiste à ajouter des serveurs supplémentaires (de manière automatique ou manuelle, définitive ou temporaire) pour augmenter les capacités et ressources disponibles

51. Une partition de réseau fait référence à la défaillance d'un périphérique réseau qui entraîne la division d'un réseau. Étant donné que des applications différentes ont des exigences différentes en matière de cohérence et peuvent tolérer l'indisponibilité dans une mesure différente, différentes stratégies de gestion des partitions sont disponibles.

NFA	déploiement	Partitions Réseau
Poids	0.8	0.2
Impact	+ : MultiAZ = : MonoAZ	+ : Mode Pause minority = : Mode autoheal - : Mode ignore

TABLE 5.15 – Résultats de l'évaluation des NFA en termes de disponibilité et de leurs impacts

élaboré un plan de référence qui servira de base de comparaison pour les services proposés par notre approche. Ce plan de référence inclut les meilleures valeurs des NFR existantes parmi les plans de services que nous avons évalués. De cette manière, nous pouvons situer les plans de services obtenus par notre solution par rapport aux meilleures valeurs des NFR obtenues dans l'ensemble de la liste des plans de services évalués. La figure 5.6 illustre les configurations de ce plan référence.

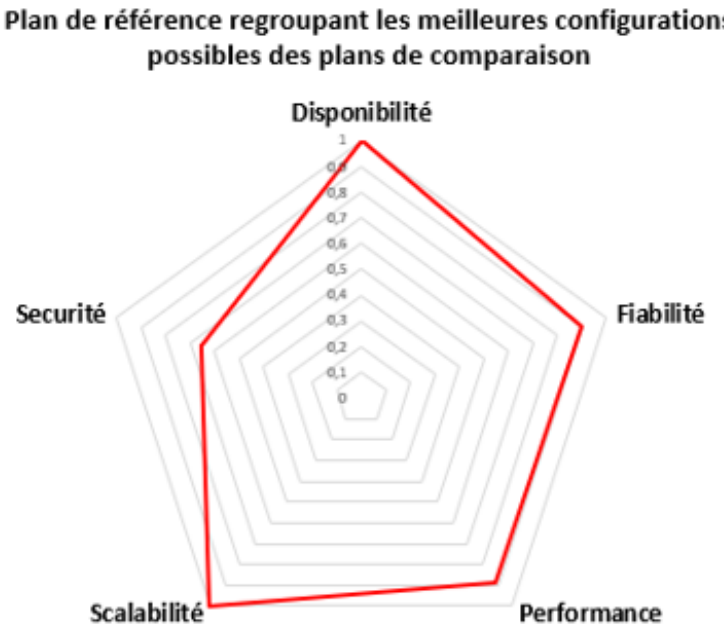

FIGURE 5.6 – Plan de référence pour les services de file d'attente

Les résultats sont validés par les architectes sur la base de la configuration des plans, qui sont présentés dans le tableau 5.10. Le service du besoin 1 nécessite une disponibilité et une fiabilité moyennes, des performances, une scalabilité et une sécurité élevées. Le coût de déploiement du service ne doit pas dépasser 4000 dollars. La figure 5.4 montre le service proposé par notre solution. La comparaison des configurations de notre solution par rapport au plan de référence (le plan de qui inclut les meilleures valeurs des NFR existantes

parmi les plans de services que nous avons évalués.) présenté dans la figure 5.6 révèle que cette solution répond parfaitement aux besoins mentionnés ci-dessus. Le plan de service *cloudamqp- RabbitMQ "Roaring Rabbit - 1 node"* offre des performances élevées avec un haut débit de 20000 messages/seconde et une faible latence de 66 ms. Il offre également une sécurité élevée avec un VPC et un chiffrement des données persistantes ainsi qu'une scalabilité verticale. En termes de disponibilité et de fiabilité, la configuration du plan a été meilleure que ce que nous attendions. Le plan fournit un déploiement multi-AZ, une persistance des données et un système ACK pour seulement $299 par mois. Une solution alternative pour le besoin 1 est présentée dans la figure 5.5 avec un prix de 3499 $. Cette solution a été choisie comme alternative au regard de son prix élevé par rapport à la solution primaire qui coûte 299 $. Comme on peut le voir sur la figure 5.5 et la figure 5.4, cette solution offre des performances légèrement supérieures à celles de la solution primaire. Cependant, les architectes préfèrent perdre un peu de performance pour le prix du déploiement.

Le service pour le besoin 2 nécessite une fiabilité et une disponibilité élevées, une sécurité, des performances et une scalabilité relativement faibles. Le coût de déploiement du service ne doit pas dépasser 4000 dollars. La figure 5.4 montre le service proposé par notre solution. Le plan de service *AMAZON : SQS FIFO* offre une haute fiabilité et une haute disponibilité avec un déploiement Multi-AZ, une persistance des messages et une garantie de réception des messages grâce à des accusés de réception (ACK). En outre, contre nos attentes, ce plan offre une scalabilité verticale et horizontale au prix de 5 dollars par mois (sans compter le coût supplémentaire des autres services lors de l'utilisation d'Amazon). en comparant les configurations de ce plan selon les besoins décrit plus haut par rapport au plan de référence 5.6 (le plan qui inclut les meilleures valeurs des NFR existantes parmi les plans de services que nous avons évalués.) nous estimons que nous fournissons une solution pertinente de comparaison des services. Une solution alternative pour le besoin 2 est présentée dans la figure 5.5 avec un prix de 1999 $. Cette solution a été choisie comme alternative d'abord au regard de son prix élevé par rapport à la solution primaire qui coûte 5 $. Ensuite, bien que cette solution alternative soit bien meilleure en termes de performances que la solution primaire, elle reste légèrement moins bonne en termes de fiabilité (qui est le besoin principal des architectes) et moins bonne en termes de scalabilité. Ces raisons expliquent pourquoi les architectes l'ont considéré comme une

solution alternative.

Dans cette sous-section, nous avons sélectionné les meilleurs services de file d'attente qui répondent aux exigences spécifiées par les architectes dans le tableau 5.10. Nous avons validé le choix approprié des services en collaboration avec les architectes sur la base de la configuration des plans.

5.4.3 Conclusion

Dans ce chapitre, nous avons présenté une méthodologie permettant de comparer les plans des fournisseurs de services ayant des exigences fonctionnelles similaires. Cette méthodologie permet de sélectionner les services appropriés pour une architecture d'application. Elle est basée sur les résultats des éléments clés de comparaison et de leurs relations obtenues dans le chapitre précédent. Nous avons d'abord présenté des définitions formelles pour modéliser notre problème de sélection de services. Ensuite, nous avons présenté notre méthodologie de sélection de services, qui comprend trois phases principales : (1) mener une enquête pour attribuer un poids significatif aux NFA et aux NFR en fonction des besoins des architectes en matière d'application ; (2) résoudre les difficultés liées à l'incapacité de mesurer l'impact des NFA sur la performance ; (3) proposer un modèle de sélection des services. Ce modèle est basé sur (i) des méthodes de normalisation des valeurs des NFA en fonction de leurs types de données ; (ii) l'utilisation de la méthode d'aide à la décision OWA pour classer les services cloud en fonction des besoins des architectes ; (iii) la proposition d'un algorithme de sélection de services qui utilise les valeurs normalisées des NFA et la méthode d'aide à la décision OWA. Pour conclure, nous avons validé notre approche de sélection des services cloud sur une base de données SQL et un service de mise en file d'attente. Cette approche a été adoptée pour des capacités et des besoins de services différents.

Nous soulignons que l'approche proposée dans cette étude peut être réutilisée pour tout type de service. Cependant, elle reste dépendante de la disponibilité des données (NFA) communiquées par les fournisseurs de services et des benchmarks disponibles dans la littérature. Elle dépend également de la disponibilité d'architectes experts du domaine pour attribuer des poids significatifs aux NFA et aux NFR en fonction de leurs besoins applicatifs.

Dans le prochain chapitre, nous présentons notre approche de composition de services

cloud.

Chapitre 6

Composition des services cloud

6.1 Introduction

Dans le chapitre précédent, nous avons abordé la deuxième problématique, qui consiste à résoudre le problème de sélection de service cloud en comparant les plans des fournisseurs de services ayant des exigences fonctionnelles similaires. Nous allons nous intéresser dans le présent chapitre à l'évaluation de la composition des services cloud.

La sélection des compositions de services cloud appropriées est un sujet d'actualité [142] [143] [144]. Les chercheurs ont appliqué plusieurs techniques et méthodes pour résoudre le problème de la composition des services cloud. Les méthodes de composition de services existantes sont pour la plupart capables d'obtenir des solutions optimales [101, 102, 104, 111] et également capables de générer des services composés dans un temps d'exécution raisonnable [98, 101, 102]. Cependant, peu de travaux abordent la question de la qualité du déploiement des services associée à la composition de services. De plus, la plupart des approches qui abordent cette question sont basées sur la simulation et la génération aléatoire des valeurs de QoS qui peuvent biaiser l'évaluation de la composition des services.

Par ailleurs, on voit une augmentation de la complexité des architectures des applications qui sont déployées avec la montée en puissance des architectures de type micro-services. Elles multiplient le nombre d'éléments à mettre en oeuvre et la diversité des choix à réaliser. Cette complexité est rendue gérable par la généralisation de la virtualisation et par les techniques d'automatisation de la gestion du déploiement et de l'administration.

L'ambition de pouvoir déployer des applications à l'architecture complexe pour différents scénarios d'utilisation et de charge sans avoir à se préoccuper de l'administration de

middleware est difficile à atteindre. Trouver la bonne composition de services et de plans pour ces services pour assurer un niveau de service suffisant pour une application donnée peut prendre beaucoup de temps. Ainsi, l'objectif de ce chapitre est de proposer une méthode pour guider les architectes dans la sélection des compositions des plans de service pour leurs architectures. Cette méthode doit leur permettre d'adapter la sélection à leurs besoins tout en contrôlant le coût opérationnel.

Ce chapitre est organisé comme suit : la section 2 décrit l'analyse du processus de composition de services. La section 3 introduit les définitions formelles pour modéliser notre problème. La section 4 décrit la méthodologie proposée pour la sélection de la composition des services cloud et donne un aperçu du processus proposé pour assurer cette composition.

6.2 Analyse du processus de composition des services

Les architectes sont souvent amenés à construire différentes architectures techniques pour leurs applications en réutilisant les mêmes services et les mêmes exigences pour les services qui composent leur architecture. La recherche d'un moyen de réutiliser ces services sans être obligé de tout refaire leur permet de bénéficier d'un environnement agile qui leur permet de développer, tester et lancer rapidement des applications. Pour atteindre cet objectif, nous proposons d'utiliser un ensemble de services cloud regroupés dans des micro-services. Chacun de ces micro-services ont des exigences non fonctionnelles au sein de l'architecture. Ces exigences varient en fonction du rôle occupé par le micro-service au sein de l'architecture technique. Ces micro-services sont appelés des templates. Un exemple d'architecture composée de 3 templates est présenté dans la figure 5.2.

Nous avons considéré qu'une architecture se compose d'un ensemble de templates et d'un ensemble de contraintes générales. Dans un premier temps, il faut spécifier les contraintes générales qui sont les contraintes que tous les services de l'architecture doivent respecter. Par exemple, il y'a la contrainte relative au coût où il est spécifié que le prix de tous les services composant l'architecture ne doit pas dépasser une somme donnée ou encore, la contrainte relative à la région où il est spécifié que tous les services doivent être déployés dans la même région pour des raisons juridiques et réglementaires.

Dans un deuxième temps, nous avons considéré que chaque template a un rôle spécifique dans l'architecture. Par conséquent, les exigences d'un template sont susceptibles

d'être différentes de celles des autres templates de la même architecture. Par exemple, un template qui contient des informations sensibles doit répondre à des exigences de sécurité élevées et les autres exigences sont de moindre priorité. Pour un autre template de la même architecture qui est utilisé pour récupérer des informations moins sensibles (par exemple les produits mis à disposition par l'entreprise), la priorité est donnée aux performances et à la disponibilité car il doit répondre assez rapidement à la demande des utilisateurs tandis que les autres exigences sont moins prioritaires.

Dans une dernière étape, d'autres contraintes de services peuvent être spécifiées. Par exemple, il peut être primordial de disposer d'une capacité de stockage de 64 Go pour un service de stockage et de 1 000 000 messages par mois pour un service de messages FIFO.

Cette étude nous a permis d'identifier les principaux objectifs à atteindre afin de fournir aux architectes une solution de composition de services adaptée à leurs besoins. Dans la prochaine section, nous présentons la formalisation du processus de composition de services.

6.3 Formalisation de la composition des services

Dans cette sous-section, nous présentons les concepts de base de la sélection de la composition des services cloud et leurs définitions formelles. Ce travail fait suite à celui sur les concepts de base de la sélection des services cloud (décrit dans la section 5.2).

6.3.1 Défintion d'un template

Soit T l'ensemble de templates existants. Un template $t \in T$ est défini comme suit :

$t = (\{s_i\} \mid i \in [1, n])$ avec :

— $\{s_i\} \subseteq S$ est l'ensemble des services dans le template t.

Exemple

Un exemple d'un template "Template 1" est présenté dans la Fig.6.1, où s_1 = Relational database, s_2= Caching.

6.3.2 Besoins des architectes en ce qui concerne les NFR par template

Soit $t \in T$ un template et soit V_{nfr} le vecteur des NFR de dimension k.

— $W_t = (w_1, ..., w_k)$ est un vecteur de pondération de dimension k, pour le template t, tel que :

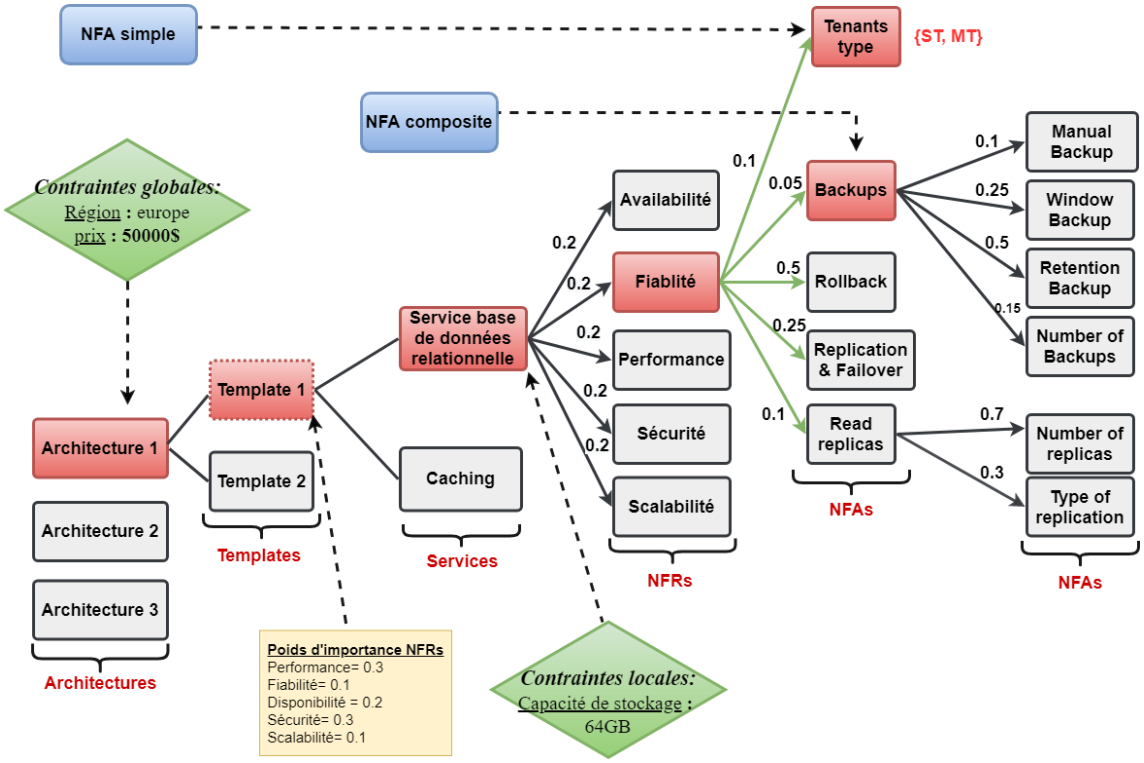

FIGURE 6.1 – Exemple Architecture, templates et exigences NFR des templates

— $w_j \in [0, 1]$

— $\sum_{j=1}^{k} w_j = 1$

Chaque w_j est donné par les architectes et dénote le poids d'importance d'un $NFR \in V_{nfr}$.

Exemple (voir Figure 6.1)

Pour le "Template 1" :

Le vecteur V_{nfr} =($Performance$, $Fiabilité$, $Disponibilité$, $Sécurité$, $Scalabilité$).

Le vecteur de pondération pour l'ensemble de NFR est respectivement comme suit :

$W_t = (0.3, 0.1, 0.2, 0.3, 0.1)$.

6.3.3 Définition d'une Architecture

Soit A l'ensemble des architectures. Soit $a \in A$ une architecture définie comme suit :

$a = (\{t_j\}, C_g \mid j \in [1, k]))$

— $\{t_j\} \subseteq T$ est l'ensemble de templates composant a.

— C_g est l'ensemble des contraintes globales sur l'architecture. Une contrainte globale peut-être par exemple le coût maximal de l'architecture et la région.

Exemple

Un exemple d'une architecture "Architecture 1" est présenté dans la Figure 6.1, Où t_1 , t_2 sont respectivement "Template 1" et "Template 2" et $C_g = (Région , coût)$, avec Région="europe" et coût = 50000$.

6.3.4 Définition d'une instanciation d'un template

Soit I l'ensemble des instanciations possibles des templates.

Soit $t \in T$ un template et $I_v \in I$ la $v^{\text{ième}}$ instance possible pour le template t. I_v est défini comme suit :

$I_v = (\{p_{im}\}, F_I, C_I, \mid i \in [1, n], m \in [1, z_i])$ avec n est le nombre des services dans t et z_i est le nombre des plans dans chaque service (z_i dépend du service).

— $\{p_{im}\}$ est l'ensemble des plans qui composent I_v. p_{im} est le $m^{\text{ième}}$ plan pour le service s_i dans t, tel qu'on ait un plan pour chaque service composant le template.

— $F_I : I \rightarrow R$: Une fonction qui associe à chaque instance de template une valeur dans R, que nous appelons score de template. Cette valeur est calculée en utilisant la formule de la définition 6.3.6

— $C_I : I \rightarrow R$: Une fonction qui associe à chaque instance de template un coût. Ce coût est calculé en utilisant la formule introduite dans la définition 6.3.7

Exemple

Dans la Figure 6.1 et la Table 6.1 :

t_1 correspond à "Template 1", avec :

t_1=($\{s_1, s_2\}$), s_1 = Relational databases et s_2 = Caching

On suppose que :

le service s_1 dans t_1 (Relational database) a 3 plans : p_{11}, p_{12}, p_{13}

le service s_2 dans t_1 (Caching) a 2 plans : p_{21}, p_{22}

I_1 l'instanciation d'indice 1 de t_1 est composé de deux plans ($\{p_{11}, p_{21}\}$).

$F_I(I_1) = 0.88$ (voir Tableau 6.1)

$C_I(I_1) = 3790$ (voir Tableau 6.1)

	Services obtenus et leurs prix	Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	Score template	Prix du template
template besoin 1	1- ServiceSQL :Amazon - db.r3.4xlarge (Provisioned IOPS - multiAZ) **3291 $** 2- cloudamqp- RabbitMQ - Power Panda - 3 nodes - multiAZ **499 $**	1.0	0.906	0.481	0.75	0.832	0.88	3790 $

TABLE 6.1 – Exemple d'une instance d'un template

6.3.5 Vecteur de valeurs de NFR pour une instance de template

Soit I_v une instance d'un template $t \in T$ composé de n plans. Soit $V_{p_{im}}$ le vecteur de valeurs de NFR de dimension k pour chaque plan p_{im} composant I_v et V_{nfr} le vecteur des NFR de dimension k.

Soit V_{I_v} le vecteur de valeurs de NFR de dimension k pour l'instance I_v.

Le $j^{\text{ième}}$ NFR noté $V_{I_v}[j]$ pour l'instance I_v est calculé comme suit :

$V_{I_v}[j] = \frac{1}{n} * \sum_{i=1}^{n} \sum_{m=1}^{z_i} \mu_{im} * V_{p_{im}}[j]$

avec :

$$\mu_{im} = \begin{cases} 1 & \text{si } p_{im} \text{ est sélectionné} \\ 0 & \text{sinon.} \end{cases}$$

et $V_{p_{im}}[j]$ est le $j^{\text{ième}}$ NFR de p_{im}.

Exemple

la Table 6.1 présente un exemple de vecteur de valeurs de NFR pour l'instance I_1 de template t_1. $V_{I_1} = (1.0, 0.906, 0.481, 0.75, 0.832)$

6.3.6 Score d'une instance d'un template

Soit I_v une instance d'un template $t \in T$. Soit W_t le vecteur de pondération pour le template t et soit V_{I_v} le vecteur de valeur de NFR pour I_v.

Le score $score_v$ associé à I_v est calculé en utilisant la méthode OWA.

$score_v = OWA(V_{I_v}, W_t)$

6.3.7 Coût d'une instance d'un template

Soit I_v une instance d'un template $t \in T$. Le coût $cost_v$ associé à I_v est calculé comme suit :

$cost_v = \sum_{i=1}^{n} \sum_{m=1}^{z_i} \mu_{im} * cost_{p_{im}}$ avec $cost_{p_{im}}$ le coût de $m^{\text{ième}}$ plan pour le service s_i et $\mu_{im} \in \{0, 1\}$.

— $\mu_{im} = 1$ si p_{im} est le plan sélectionné pour le service s_i pour I_v.

— $\mu_{im} = 0$ dans le cas inverse.

6.3.8 Définition d'une instanciation d'une architecture

Une instance d'architecture est un déploiement possible tel qu'on ait une instance pour chaque template composant l'architecture.

Soit A_r l'ensemble d'architecture résultat qui correspond à l'architecture $a \in A$.

Soit $a_k \in A_r$ la $k^{\text{ième}}$ architecture résultat pour a. L'instance a_k est défini comme suit :

$a_k = (\{I_{jv}\}, F_a, C_a, \mid j \in [1, d], v \in [1, e_j])$, avec d le nombre des templates dans l'architecture a et e_j est le nombre des instances d'un template (e_j dépend de la composition de template en terme de plans).

— $\{I_{jv}\}$ représente l'ensemble des templates qui composent a_k et I_{jv} est la $v^{\text{ième}}$ instance de template t_j dans a, tel qu'on ait une instance pour chaque template composant l'architecture.

— $F_a : A_r \rightarrow R$ est une fonction qui associe à chaque architecture résultat a_k une valeur dans R que nous appelons le score d'une architecture résultat. Ce score est calculé en utilisant la formule 6.3.9.

— $C_a : A_r \rightarrow R$ est une fonction qui associe à chaque architecture résultat un coût. Ce coût est calculé en utilisant la formule intriduite dans 6.3.10

6.3.9 Score d'une architecture

Soit $a_k \in A_r$ une architecture résultat. Supposons que a_k est composé de d instances de template. Le score $Score_{a_k}$ associé à a_k est calculé comme suit :

$Score_{a_k} = \frac{\sum_{j=1}^{d} \sum_{v=1}^{e_j} \mu_{jv} * Score_{jv}}{d}$, avec j est l'indice sur les templates et v est l'indice sur les instances d'un template.

— $Score_{jv}$ est le score de la $v^{\text{ième}}$ instance du template t_j dans l'architecture a (voir

Définition 6.3.4).

$$\mu_{jv} = \begin{cases} 1 & \text{si } I_{jv} \text{ est sélectionné} \\ 0 & \text{sinon.} \end{cases}$$

— d est le nombre des templates qui composent l'architecture a_k.

6.3.10 Coût d'une architecture résultat

Soit a_k une architecture résultat pour $a \in A$. Le coût $cost_{a_k}$ associé à a_k est calculé comme suit :

$cost_{a_k} = \sum_{j=1}^{d} \sum_{v=1}^{e_j} \mu_{jv} * cost_{jv}$

Avec $cost_{jv}$ le coût d'une instance v d'un template t_j (Voir 6.3.7) et $\mu_{jv} \in \{0, 1\}$.

$$mu_{jv} = \begin{cases} 1 & \text{si } I_{jv} \text{ est sélectionné} \\ 0 & \text{sinon.} \end{cases}$$

6.4 Méthode proposée

Notre objectif principal est de proposer une méthode pour guider les architectes dans la sélection des compositions de plans de services pour leurs architectures. Elle doit permettre aux architectes d'adapter la sélection à leurs besoins tout en contrôlant le coût opérationnel. De plus, la méthode doit être capable de générer des compositions de plans dans un temps raisonnable. Pour atteindre cet objectif, les architectes doivent fournir en entrée un ensemble de services qu'ils souhaitent déployer pour chaque template. Ensuite, ils doivent spécifier leurs exigences pour les services en termes de NFR (comme expliqué dans la section précédente) et exprimer leurs exigences en termes de NFR pour chacun des templates composant l'architecture. Enfin, ils doivent exprimer leurs contraintes globales pour l'architecture, par exemple le coût maximum à ne pas dépasser pour tous les services composant l'architecture et/ou la zone dans laquelle tous les services composant l'architecture doivent être situés.

Dans notre approche, la composition de services permet la création de plusieurs instances de templates où chaque instance de template est composée de plusieurs instances de services (plans). Au fur et à mesure que nous progressons dans le processus de composition de services, nous rencontrons plusieurs solutions potentielles qui prendraient un temps exponentiel pour traiter toutes les solutions. Par exemple, s'il existe m services abstraits pour

un template et n services candidats (plans), le nombre de compositions de services possibles pour ce template est n^m [145]. Comme il s'agit d'un problème NP-Complet, nous devons trouver de bonnes solutions qui répondent à notre objectif dans un temps raisonnable.

Les solutions les plus pertinentes peuvent être trouvées grâce à des algorithmes très efficaces, souvent appelés métaheuristiques [146]. Ces méthodes de recherche sont fortement recommandées pour obtenir de bonnes solutions, qui sont les plus pertinentes et peuvent l'être moins dans certains cas, en temps polynomial au lieu du temps exponentiel qui se produit lorsque nous résolvons ces problèmes en utilisant des méthodes conventionnelles.

Dans la présente section, nous proposons un algorithme génétique pour sélectionner les compositions de services les plus appropriées dans un temps d'exécution raisonnable.

6.4.1 Présentation de l'algorithme génétique et de son fonctionnement pour la composition

Les algorithmes qui utilisent des méthodes exactes deviennent inutilisables quand le nombre de templates, le nombre de services et le nombre de plans de services deviennent trop importants. Il est impossible de parcourir et de traiter toutes les combinaisons existantes en un temps raisonnable. Dans ces cas, l'utilisation d'une métaheuristique telle que l'Algorithme Génétique (AG) devient indispensable pour obtenir une réponse satisfaisante en un temps raisonnable.

Les AG [147] sont des approches heuristiques qui trouvent de manière itérative les solutions les plus appropriées dans de grands espaces de recherche. Toute solution possible au problème d'optimisation est codée sous la forme d'un chromosome (génome). Un ensemble de chromosomes est appelé une population. La première étape d'un AG consiste à dériver une population initiale. Un ensemble aléatoire de chromosomes est souvent utilisé comme population initiale. Cette population initiale est la première génération à partir de laquelle l'évolution commence. La deuxième étape est le processus de sélection, chaque chromosome est éliminé ou dupliqué (une ou plusieurs fois) en fonction de sa qualité relative. La taille de la population est généralement maintenue constante. L'étape suivante est le processus de croisement. Certaines paires de chromosomes sont sélectionnées dans la population actuelle et certains de leurs composants correspondants sont échangés pour former deux chromosomes valides. Après le croisement, chaque chromosome de la population peut subir une mutation avec une certaine probabilité. Le processus de mutation transforme un chromo-

some en un autre chromosome valide. La nouvelle population est alors évaluée et chaque chromosome est associé à une valeur d'aptitude, qui est une valeur obtenue à partir de la fonction objective. L'objectif de l'évaluation est de trouver un chromosome qui a la valeur d'aptitude optimale. Si le critère d'arrêt n'est pas rempli, la nouvelle population passe par un autre cycle (itération) de sélection, de croisement, de mutation et d'évaluation. Ces cycles se poursuivent jusqu'à ce que le critère d'arrêt soit rempli [148].

6.4.2 Paramètres de l'algorithme génétique

L'approche que nous proposons pour la composition de services consiste à utiliser les paramètres suivants pour l'algorithme génétique.

6.4.2.1 Définir le chromosome

Pour les algorithmes génétiques, l'une des questions clés est de coder une solution du problème en un chromosome (individu ou une composition). Dans notre modèle, le chromosome est codé par un tableau binaire dont le nombre d'éléments correspond au nombre de plans converti en binaire pour chaque service. Un chromosome est représenté comme suit :

$C = (\{p_{jik}\} \mid j \in [1, d], k \in [1, m], i \in [1, n])$, avec p_{jik} est le $k^{\text{ième}}$ plan du $i^{\text{ième}}$ service dans le template t_j.

Prenons l'exemple d'une architecture composée de deux templates. Le template t_1 est composé de deux services, le service s_{11} avec 120 plans et le service s_{12} avec 90 plans. Le template t_2 est composé de deux autres services, le service s_{21} avec 300 plans et le service s_{22} avec 650 plans. Cet exemple est représenté dans la Table 6.2. Il s'agit de la représentation binaire du nombre de plans pour chaque service et du nombre de bits sur lesquels les différents plans de service peuvent être représentés.

Architecture			
t_1		t_2	
s_{11}	s_{12}	s_{12}	s_{22}
120 plans	90 plans	300 plans	650 plans
En binaire			
1111000	1011010	100101100	1010001010
Taille de chaque service			
7 bits	7 bits	9 bits	10 bits
Taille du chromosome			
7 bits + 7 bits + 9 bits + 10 bits = 33 bits			

TABLE 6.2 – Exemple du nombre de bits sur lesquels est représenté un chromosome

Nous déterminons d'abord la taille en bits du chromosome de l'individu (une seule composition de services), en trouvant le nombre de plans pour chaque service. Ensuite, nous générons la structure du chromosome qui contiendra 7+7+9+10 = 33 bits (la somme des bits représentés par chaque service). La Table 6.3 illustre un chromosome représenté par les premiers plans de chaque service.

Indice (x1) du service 1	Indice (x2) du service 2	Indice (x3) du service 3	Indice (x4) du service 4
0|0|0|0|0|0|0	0|0|0|0|0|0|0	0|0|0|0|0|0|0|0|0	0|0|0|0|0|0|0|0|0|0

TABLE 6.3 – Exemple de représentation d'un chromosome

Pour qu'un individu (chromosome ou une composition de services) soit valide, il doit répondre aux conditions suivantes :

1. chaque indice qui le compose doit appartenir à l'intervalle correspondant.
2. le prix total de l'individu (la composition), qui est la somme des prix des plans qui le composent, doit être inférieur à la contrainte de prix indiquée par les architectes pour l'ensemble de leur architecture.

6.4.2.2 Générer la population initiale

Un nombre prédéfini de chromosomes est généré pour former la génération initiale. Le chromosome ayant la meilleure valeur d'aptitude est toujours classé en premier.

6.4.2.3 Appliquer l'opérateur génétique

Pour appliquer ce processus, nous définissons l'opérateur de chaque étape comme suit :

— opérateur de sélection : nous utilisons la sélection de tournoi binaire comme opérateur de sélection. La sélection de tournoi binaire organise un tournoi entre deux individus et sélectionne le gagnant. De cette façon, les individus qui vont former la prochaine génération sont déterminés. La taille de la population de chaque génération est toujours P.

— opérateur de croisement : nous utilisons plusieurs points de croisements comme opérateur de croisement. Le nombre de points de croisements est généré selon une probabilité ´$Prob$ˇ.

— opérateur de mutations : selon la probabilité de mutation, un ou plusieurs gènes peuvent être permutés.

6.4.2.4 Évaluer les chromosomes à l'aide de la fonction de fitness

Le score de fitness est le score associé à la qualité des déploiements des templates composant l'architecture. Pour calculer ce score, les scores des templates sont additionnés et divisés par le nombre de templates. Le score de chaque template est d'abord calculé en additionnant les scores de chacun des NFR des services appartenant au même template. Par exemple, dans la figure 6.1 et pour le NFR de performance, nous additionnons les scores de performance du service de file de messages et du service de base de données relationnelle. Le résultat du score de performance pour le template est présenté à l'étape 1 de la figure 6.2. Il en résulte un score pour chaque NFR qui compose le template. Ensuite, en utilisant la méthode OWA avec les vecteurs de poids d'importance des NFR des templates, on calcule le score de fitness qui correspond à un ensemble de plans composant chacun des templates tout en respectant la contrainte de prix (le prix des plans composant l'architecture ne doit pas dépasser un prix seuil spécifié par les architectes).

6.4.3 Condition d'arrêt

L'algorithme génétique s'arrête dans deux situations données. La première est celle où le nombre de générations défini pour l'algorithme est atteint. La seconde est celle où le score acceptable pour une solution satisfaisante est atteint. Avec un nombre illimité de générations, la meilleure valeur de score d'un template pour notre algorithme est égale à 1. Ainsi, la condition d'arrêt pour un seul template consiste à ce que la génération atteigne un score de 1. Pour deux templates, la génération doit atteindre un score de 2. Pour trois

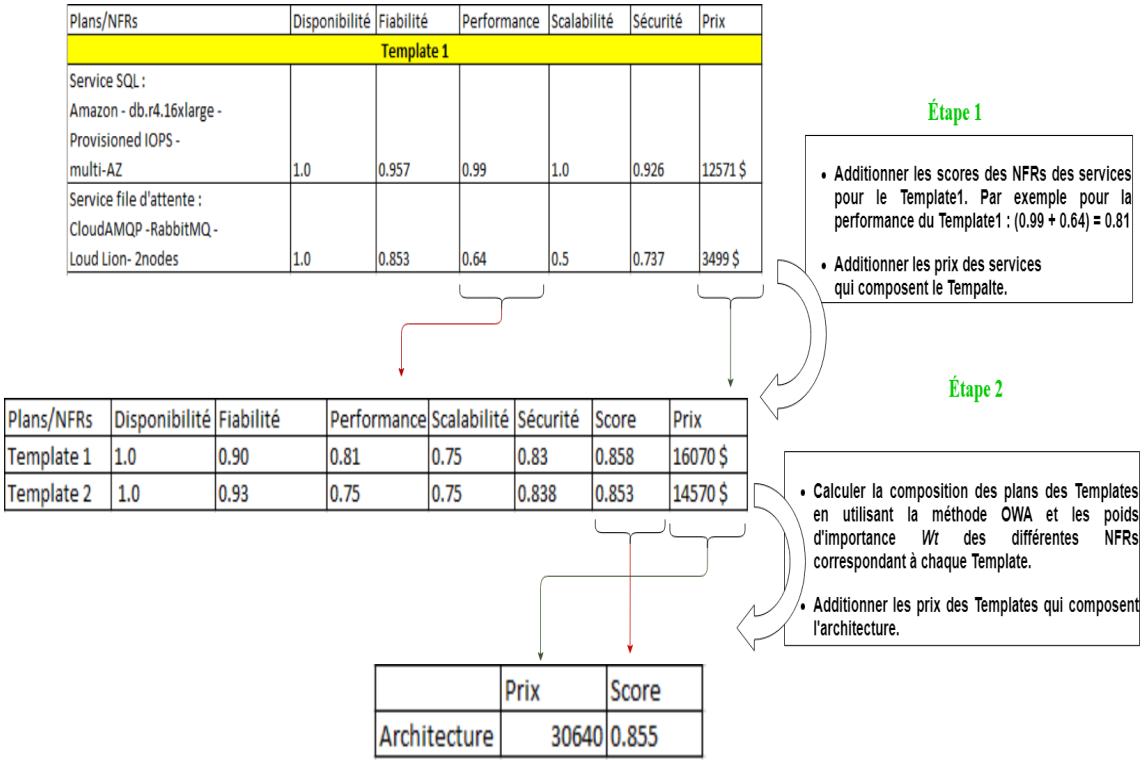

FIGURE 6.2 – Les différentes étapes pour le calcul du score de fitness pour l'architecture

templates, la génération doit atteindre un score de 3. Il est possible de modifier cette condition d'arrêt pour atteindre une génération avec un score déterminé (score d'arrêt souhaité pour un template multiplié par le nombre de templates). Pour y parvenir, il est nécessaire de réaliser des tests afin de déterminer à quel score de template on souhaite s'arrêter pour arriver à une solution satisfaisante.

6.4.4 Aperçu du processus de composition des services

La figure 6.3 présente notre processus de la composition des services. La description des variables et des fonctions de ce processus sont expliquées respectivement dans les tableaux 6.4 et 6.5

Ce processus prend en entrée une architecture, la région où les services qui composent l'architecture doivent être situés, le prix maximum de l'architecture qui ne doit pas être dépassé, le score à considérer comme une solution acceptable pour arrêter l'algorithme génétique et les paramètres de l'algorithme génétique. Le résultat de ce processus est un ensemble de plans de service pour chaque template selon les exigences NFR exprimées par les architectes.

Ce processus sélectionne d'abord tous les plans de services qui se trouvent dans la

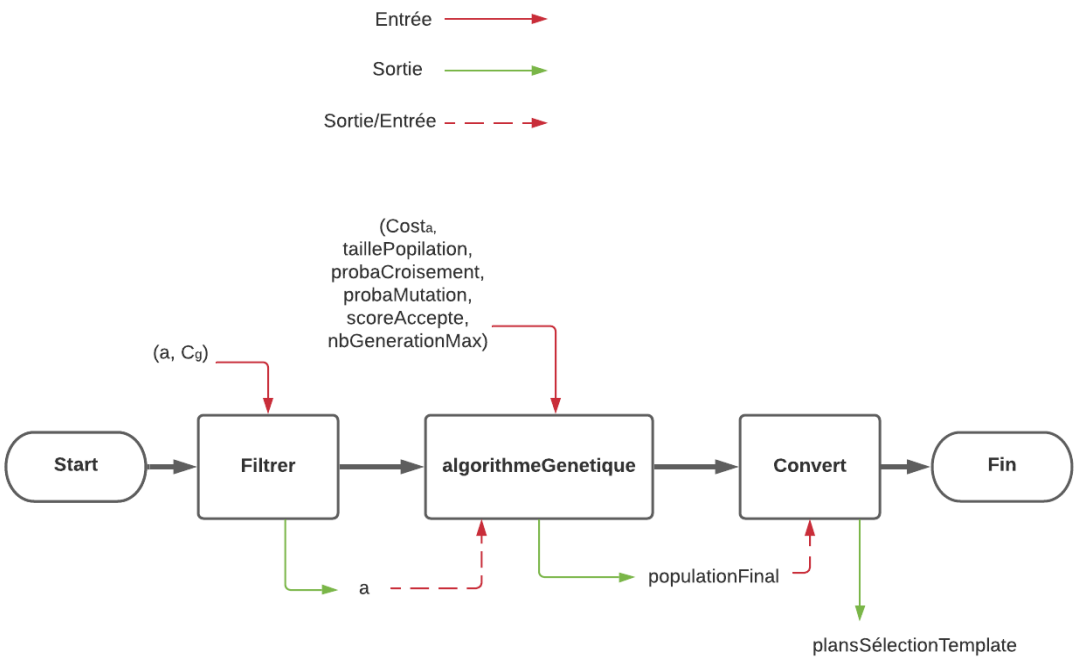

FIGURE 6.3 – Processus de sélection de la composition de services

région saisie (étape 1). Ensuite, il fait appel à l'algorithme génétique décrit dans la section 6.4.2, auquel sont fournis le prix maximum que les services composant l'architecture ne doivent pas dépasser, le score pour lequel l'algorithme génétique doit s'arrêter, et les autres paramètres décrits dans cet algorithme génétique. L'algorithme génétique renvoie une liste d'individus (une liste de compositions de plans des services) par template, considérés comme les meilleures solutions pour l'architecture en fonction des besoins exprimés par les architectes (étape 2). Enfin, comme cette liste d'individus est codée en binaire, elle est décodée et proposée comme une liste de plans pour chaque template afin que les architectes puissent les exploiter (étape 3).

6.4.5 Synthèse

Dans cette section, nous avons proposé une approche de sélection de la composition des services cloud. Cette approche se base sur les résultats de la sélection de services. Nous pouvons séparer notre étude en trois parties, la première consiste à identifier les principaux objectifs à atteindre afin de fournir aux architectes une solution de composition de services adaptée à leurs besoins. La deuxième partie consiste à formaliser les besoins des architectes concernant la composition de services. La troisième partie consiste à proposer un processus

Nom des Variables	Type	Description
a	Structure	Une architecture a est composée de plusieurs templates $\{t_i\}$. Chaque template t_i est composé d'un ensemble de services $\{s_j\}$ et possède un vecteur de poids d'importance W_{t_i} pour les NFR. Chaque service s_j est composé d'un ensemble de plans $\{p_k\}$ et possède un vecteur de poids d'importance W_{s_j} pour les NFR.
C_g	String	Les contraintes que l'architecture doit respecter. Par exemple, la région où doivent être situés les services qui composent l'architecture.
$cost_a$	Double	Le coût maximum que la somme des services qui composent l'architecture ne doit pas dépasser.
taillePopulation	int	La taille de la population gérée par l'algorithme génétique.
probaCroisement	Décimal	La probabilité de croisement utilisée par l'algorithme génétique.
probaMutation	Décimal	La probabilité de mutation utilisée par l'algorithme génétique.
scoreAccepte	Décimal	Le score à considérer comme une solution acceptée pour arrêter l'algorithme génétique.
nbGenerationMax	Décimal	Le nombre de générations maximal (l'une des conditions d'arrêt de l'algorithme génétique).
plansSélectionTemplate	Liste des vecteurs	Liste des plans de service codé en binaire qui représentent les combinaisons des templates sélectionnés pour l'architecture.
populationFinal	Liste des individus	Ensemble de plans de services pour chaque template en fonction des besoins NFR exprimés par les architectes pour chacun des templates.

TABLE 6.4 – Spécification des variables du processus de composition des services 6.3

	Entrées	Sorties	Description
Filtre	- Identifiant architecture - C_g	liste de plans répondant aux contraintes globales de l'architecture par exemple celle de la région.	Filtre la liste des plans de chaque template. service de l'architecture fournie en entrée.
AlgorithmeGenetique	- Identifiant architecture - $cost_a$ - taillePopulation - probaCroisement - probaMutation - scoreAccepte - nbGenerationMax	Liste d'individus	Voir la description de l'algorithme génétique plus bas.
convert	- Liste d'individus	Liste des vecteurs	Convertie la liste des individus (binaire) à une liste de vecteurs (plans).

TABLE 6.5 – Spécification des fonctions du processus de composition des services 6.3

de composition de services qui permet de générer des solutions de composition dans un temps raisonnable, tout en adaptant la sélection des plans de services en fonction des besoins des architectes, notamment en termes de coûts.

Dans la prochaine section, nous présentons la validation de notre approche de composition de services pour différents besoins non fonctionnels à différents coûts. Nous l'appliquons à la sélection de services cloud composés de deux types de services : base de données relationnelle et services de file de messages.

6.5 Validation de composition des services

Nous validons la pertinence de notre approche de composition de services en utilisant deux méthodes différentes :

— La première consiste à évaluer chaque template séparément. À cet égard, nous utilisons différentes exigences non fonctionnelles pour les templates et différentes exigences non fonctionnelles pour les services constituant ces templates. Nous utilisons également deux contraintes. La première contrainte est que les services constituant les templates doivent être basés dans une région spécifique, par exemple en Europe. La deuxième contrainte est que le coût global des services constituant le template ne doit pas dépasser un montant prédéterminé.

— La deuxième méthode de validation consiste à valider des architectures composées de plusieurs templates. Pour ce faire, nous réutilisons chacune des exigences des templates de la première méthode pour construire des architectures techniques. La contrainte à respecter est que le coût global de l'ensemble des templates constituant l'architecture ne doit pas dépasser un montant prédéterminé.

6.5.1 Validation des choix de services pour les templates

Pour valider notre approche, nous l'avons appliquée à la sélection de templates de services cloud composée de deux types de services : les services de base de données SQL et services cloud file de messages. Nous avons choisi ces deux services en raison de leur utilisation fréquente par les architectes lors du déploiement de leur architecture technique. Pour le service de base de données SQL, nous avons considéré les exigences mentionnées dans la Table 6.6. Pour le service de files de messages, nous avons considéré les exigences

Fournisseurs	Capacité de stockage par mois	Région
Amazon et Azure	64 GB	Europe

TABLE 6.6 – Besoins des services cloud SQL pour les templates.

Fournisseurs	Capacité minimale de messages par file de messages	Nombre de files de messages	Nombre de requêtes par mois	Nombre de connexions par file de messages	Région
Amazon, Heroku et cloudAMQP	100	1000	10.000.000	100	Eurpoe

TABLE 6.7 – Besoins des services cloud files de messages pour les templates.

mentionnées dans la Table 6.7.

La Table 6.8 montre les besoins de chacun des template et de ses services. Nous avons considéré que les besoins des template sont les même que ceux des services appartenant au même template. Nous utilisons le même système de distribution de points discuté dans la section 5.3.1 pour définir les exigences de chaque template et de ses services. Nous avons 100 points à distribuer entre les NFR des différents template et des services. Ces points sont attribués en fonction des exigences de chacun des template et de chaque service comme expliqué dans la section 5.3.1. Nous définissons les besoins non fonctionnels spécifiés dans 6.8 pour les considérations suivantes :

— pour le besoin 1, nous donnons plus d'importance à la performance et à la disponibilité car ces deux NFR sont contradictoires. Par conséquent, il serait intéressant de déterminer les variations des scores de ces deux NFR simultanément.

— pour le besoin 2, nous donnons plus d'importance à la disponibilité et à la fiabilité car généralement ces deux NFR peuvent être confondues. Par conséquent, il serait intéressant de déterminer les variations des scores de ces deux NFR en même temps.

— pour le besoin 3, nous donnons plus d'importance à la scalabilité et à la sécurité. Ainsi, nous pouvons déterminer les variations des scores de ces deux NFR en fonction du prix.

	Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	Total points
Besoin 1 du template et des services appartenant à ce template	35 points	10 points	35 points	10 points	10 points	100 points
Besoin 2 du template et des services appartenant à ce template	35 points	35 points	10 points	10 points	10 points	100 points
Besoin 3 du template et des services appartenant à ce template	10 points	10 points	10 points	35 points	35 points	100 points

TABLE 6.8 – les exigences non fonctionnelles des services et des templates

Nous avons élaboré 3 grands scénarios pour comparer le même template avec les mêmes services sous différentes exigences non fonctionnelles et différentes contraintes de coûts. En

effet, un scénario consiste à fixer le coût global du template et à modifier les besoins à chaque fois, ce qui fait qu'au total, nous avons 9 scénarios. Par exemple, notre premier scénario consiste à fixer le coût à 3960 $. Pour ce scénario, nous donnons les solutions obtenues ainsi que leurs alternatives pour les trois besoins présentés dans la Table 6.8.

6.5.1.1 Sélection de templates pour un coût de 3960 $

La première étape de notre méthodologie consiste à utiliser les exigences non fonctionnelles représentées dans la Table 6.8, pour classer les templates de services. Dans la table 6.15, nous présentons le résultat de la première étape, qui comprend les solutions de base et les alternatives obtenues pour chacune des exigences. En outre, il présente la configuration des plans de services obtenue pour les différents templates.

Besoin 1

Le premier besoin du template requiert, d'une part, une performance (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une fiabilité (10 pts) et une scalabilité (10 pts) moins importantes, le tout pour un coût de 3960$. Comme nous l'avons mentionné précédemment, le template est composé de deux types de services : (1) un service de base de données SQL et (2) un service cloud file de messages.

— *Pour le service de base de données SQL* : comme le montre la Table 6.15, le meilleur plan de service SQL en termes de score de performance est le "ServiceSQL : Amazon - db.r3.8xlarge (Provisioned IOPS - monoAZ)" avec un score de 0.879 contre un score de disponibilité égale à 0.48. Ce plan coûte 3167$. En outre, parmi les meilleurs scores de disponibilité du service SQL figure le plan "ServiceSQL : Amazon - db.r3.4xlarge (Provisioned IOPS - monoAZ)" avec un score maximal de disponibilité égal à 1 et un score de performance élevée de 0.58. Ce plan coûte 3291 $. Ce dernier plan a été sélectionné par notre solution en raison de son score de disponibilité élevé et de son score de performance élevé.

— *Pour le service de file de messages* : comme le montre la Table 6.15, le meilleur plan de service correspond au plan "cloudamqp- RabbitMQ - Power Panda - 3 nodes " avec un score de performance égal à 0.40 contre un score de disponibilité égale à 1 pour un prix de 499$. Ce plan a été sélectionné par notre solution en raison de son score de disponibilité élevé et de son score de performance élevé.

— Comme le montre la Table 6.15, le coût total du template composé du plan de

		Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	prix	Score template
Besoin 1		35 pts	10 pts	35 pts	10 pts	10 pts		
Solution de Base	ServiceSQL : Amazon-db.r3.4xlarge (Provisioned IOPS -multiAZ)	1.0	0.957	0.560	1.0	0.926	3291 $	
	cloudamqp- RabbitMQ - Power Panda - 3 nodes -multiAZ	1.0	0.855	0.402	0.5	0.737	499$	
	NFR Instance template	1.0	0.906	0.481	0.75	0.832	3790$	0.767
Solution alternative	ServiceSQL : Amazon - db.r4.4xlarge (Provisioned IOPS -multiAZ)	1.0	0.957	0.553	1.0	0.926	3339$	
	FileAttente1 :SQS Standard	1.0	0.988	0.155	1.0	0.737	4$	
	NFR Instance template	1.0	0.973	0.354	1.0	0.832	3343$	0.754
Besoins 2		35 pts	35pts	10 pts	10 pts	10 pts		
Solution de Base	ServiceSQL : Amazon - db.r4.4xlarge (Provisioned IOPS -multiAZ)	1.0	0.957	0.553	1.0	0.926	3339$	
	FileAttente1 :SQS Standard∼	1.0	0.988	0.155	1.0	0.737	4$	
	NFR instance templates	1.0	0.973	0.354	1.0	0.832	3343$	0.91
Solution alternative	ServiceSQL : Amazon - db.r4.4xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.553	1.0	0.926	3339$	
	FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $	
	NFR instance templates	1.0	0.973	0.352	1.0	0.838	3344$	0.909
Besoins 3		10 pts	10 pts	10 pts	35 pts	35pts		
Solution de Base	ServiceSQL : Amazon - db.r3.8xlarge (Provisioned IOPS - monoAZ)	0.481	0.899	0.879	1.0	0.939	3167 $	
	FileAttente1 :SQS Standard∼	1.0	0.988	0.155	1.0	0.737	4$	
	NFR instance template	0.740	0.943	0.517	1.0	0.838	3172 $	0.863
Solution alternative	ServiceSQL : Amazon - db.r3.8xlarge (Provisioned IOPS - monoAZ)	0.481	0.899	0.879	1.0	0.939	3167 $	
	FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $	
	NFR instance template	0.740	0.943	0.515	1.0	0.838	31723$	0.863

TABLE 6.10 – Résultats de la solution de base et ses alternatives pour les besoins des trois templates pour un prix de 3960 $

file de messages "cloudamqp- RabbitMQ - Power Panda - 3 nœuds" et du plan SQL "Amazon - db.r3.8xlarge - Provisioned IOPS (monoAZ)" est égal à 3790 $. La meilleure valeur de score de performance pour le template que nous pouvons avoir en sélectionnant la configuration des meilleurs plans en termes de performance pour une contrainte de prix de 3960 $ est égale à (0,40 + 0,87) /2= 0,635. Cependant, en sélectionnant ce template, le score de disponibilité sera réduit (1+0.48) /2=0.74. Contrairement à la solution obtenue par notre méthode où nous avons un score de disponibilité élevé (0.40 + 0.56) /2 = 0.48 et un excellent score de disponibilité égal à 1 (1 pour la disponibilité du plan SQL et 1 pour la disponibilité du service de file de messages).

— Nous constatons que, pour un prix de 3960 $, nous obtenons une solution qui est très bonne en termes de score de disponibilité et de score de performance, comme l'exige le template 1. Cependant, avec cette adaptation, nous pouvons observer une diminution des scores obtenues pour les autres NFR. Par exemple, le meilleur score de scalabilité pour un plan de service de file de messages SQS standard qui coûte 4 $ est égal à 1. Cependant, le score de la scalabilité du plan que nous avons sélectionné pour le service de file de messages est égal à 0,5. La valeur de scalabilité obtenue pour notre template a un score de 0,75 contre 1 pour la meilleure solution en ce qui concerne la scalabilité. De même, nous remarquons une légère diminution du score de la fiabilité avec une valeur de 0,90 pour le template choisi par notre méthode contre 0,95 pour la meilleure solution en ce qui concerne la fiabilité des templates. Les résultats obtenus pour les besoins 1 du template et les différentes configurations de services sont mentionnés respectivement dans la Table 6.15.

— Il existe d'autres solutions alternatives pour l'exigence 1 du template, qui semblent intéressantes, où nous diminuons les scores de performance alors que nous améliorons les scores des autres NFR. Ce sont les solutions alternatives mentionnées dans la Table 6.15. La solution alternative pour le template d'Exigence 1 propose le Plan "ServiceSQL : Amazon - db.r4.4xlarge (Provisioned IOPS - multiAZ) " pour le service SQL à la place du Plan "Amazon - db.r3.4xlarge (Provisioned IOPS - multiAZ)" proposé par la solution de base. Ces deux plans ont exactement les mêmes configurations en termes de performances, de sécurité, de disponibilité, de scalabilité et de fiabilité. La différence réside dans le prix du plan "ServiceSQL : Amazon

- db.r4.4xlarge (Provisioned IOPS - multiAZ)" qui coûte 3339 $, donc légèrement plus cher que le plan "Amazon - db.r3.4xlarge(Provisioned IOPS - multiAZ)" qui coûte 3291 $. Quant au service de file de messages, la solution alternative propose le plan "SQS FIFO" qui coûte 5 $ à la place du plan "cloudamqp- RabbitMQ - Power Panda - 3 nodes - multiAZ" qui coûte 499 $. En comparant ces deux plans (voir la Table des services), nous remarquons que par rapport au plan de file de messages de base "cloudamqp- RabbitMQ - Power Panda - 3 nœuds - multiAZ", le score de performance diminue de 0,4 pour le service de file de messages de base correspondant à l'exigence 1 du template à 0,155 pour le service de file de messages alternatif de cette même exigence de template. En revanche, nous augmentons le score de la scalabilité de 0,5 pour le service de file de messages de base à 1 pour le service de file de messages alternatif du template. Nous améliorons également le score de la fiabilité de 0,855 pour le service de file de messages de base à 0,988 pour le service de file de messages alternatif de ce template.

Pour résumer, en comparant la solution de base avec la solution alternative mentionnée dans la Table 6.15, nous remarquons une diminution du score de performance pour la solution alternative par rapport à la solution de base. En effet, un score de performance de 0,481 pour la solution de base contre 0,354 pour la solution alternative. Cependant, nous augmentons le score de fiabilité et de scalabilité. Une augmentation du score de la scalabilité de 0,75 pour la solution de base à 1 pour la solution alternative. De même, une augmentation du score de la fiabilité de 0,906 pour la solution de base à 0,973 pour la solution alternative. Il existe d'autres alternatives intéressantes comme la sélection du plan SQL "Amazon - db.r4.4xlarge (Provisioned IOPS - multiAZ)" qui coûte 3339 $ à la place du plan "Amazon - db.r3.4xlarge (Provisioned IOPS - multiAZ)" qui coûte 3291 $ pour la solution de base. Les valeurs NFR de ces deux plans sont exactement les mêmes. En ce qui concerne le service de file de messages, nous pouvons conserver le même plan de file de messages que la solution de base "cloudamqp-RabbitMQ - Power Panda - 3 nœuds - multiAZ" qui coûte 499 $. Dans ce cas, nous aurons deux solutions qui sont identiques pour les valeurs NFR. La seule différence entre la solution de base et la solution alternative est la différence de prix qui peut être considérée comme négligeable. La solution alternative est plus chère que la solution de base de 48 $.

Besoin 2

Le deuxième besoin du template requiert, d'une part, une fiabilité (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une performance (10 pts) et une scalabilité (10 pts) moins importantes, le tout pour un coût de 3960$.

— *Pour le service de base de données SQL* : Comme le montre la Table 6.15, le plan de service SQL qui répond le mieux au besoin 2 est "ServiceSQL : Amazon -db.r4.4xlarge (Provisioned IOPS -multiAZ)" avec un score de disponibilité maximale égale à 1 et un score de fiabilité élevé égale à 0,957 pour un prix de 3343 $.

— *Pour le service de file de messages* : Comme le montre, la Table 6.15, le meilleur plan de service correspond au "SQS standard" avec un score disponibilité maximale égale à 1 et un score de fiabilité élevé égale à 0,988 pour un prix de 4$.

— En termes de fiabilité, la meilleure valeur de score que nous pouvons avoir pour le template est égale à (0.957 + 0.988) /2= 0.973. Ces résultats correspondent parfaitement aux scores de disponibilité et de fiabilité proposés par notre solution pour le template, qui sont respectivement égaux à 1 et 0.973. En effet, notre solution propose le plan SQL "ServiceSQL : Amazon -db.r4.4xlarge (Provisioned IOPS -multiAZ)" qui coûte 3343 $ pour les services SQL. Elle propose le plan de file de messages "SQS Standard" qui coûte 4 $ pour les services de file de messages.

  Nous remarquons que cette solution offre, également, pour le template un score de scalabilité élevé égale à 1 et un score de sécurité élevé égale à 0.832. Ce score de sécurité varie légèrement par rapport à la valeur maximale (0.838).

  Cependant, nous remarquons une diminution du score de la performance du template de besoin 2. Nous avons un score de performance 0.354 pour le template du besoin 2 contre un score de 0.481 pour le template de besoin 1.

— La solution alternative est de remplacer dans la solution de base le plan "SQS Standard" qui coûte 4$ par le plan "SQS FIFO" qui coûte 5 $. Ces deux plans ont exactement la même configuration sur tous les NFR à l'exception de la performance où nous baissons légèrement le score. Nous pouvons considérer cette petite différence de score comme négligeable (0.155 pour le score de la performance du plan "SQS Standard" contre "0.150" pour le score de la performance du service SQS FIFO). En effet, ces deux plans sont similaires dans leur configuration, la différence réside dans le scénario de leur utilisation. Dans l'hypothèse où nous aurions besoin d'utiliser

une file de messages FIFO, nous devrons choisir le plan "SQS FIFO". Sinon, nous pouvons utiliser les deux plans.

Besoin 3 Comme indiqué dans la Table 6.8 le besoin 3 du template exige, d'une part, une sécurité (35 pts) et une scalabilité (35 pts) plus importantes et, d'autre part, une fiabilité (10 pts), une performance (10 pts) et une disponibilité (10 pts) moins importantes, le tout pour un prix de 3960 $.

— *Pour le service de base de données SQL :* Comme le montre la Table 6.15, la meilleure valeur de sécurité qu'un service SQL peut fournir est égale à 0.939. De plus, la meilleure valeur de scalabilité que nous pouvons identifier pour le même type de service est égale à 1. Notre solution propose le plan de service "Amazon - db.r3.8xlarge (Provisioned IOPS - monoAZ) " qui répond le mieux au besoin 2 avec une valeur de sécurité égale à 0.939 et une scalabilité égale à 1 pour un prix de 3167 $.

— *Pour le service de file de messages :* Comme le montre la Table 6.15, la meilleure valeur de sécurité qu'un service de file de messages peut fournir est égale à 0.737. De plus, la meilleure valeur de scalabilité que nous pouvons identifier pour le même type de service est égale à 1. Notre solution propose le plan de service "SQS Standard" qui coûte 4 $ ayant des valeurs de sécurité et de scalabilité respectivement égales aux meilleurs valeurs de 0.737 et 1 pour les services de file de messages.

— Ainsi, en termes de sécurité, la meilleure valeur que nous pouvons avoir pour le template est égale à (0.939+0.737)/2= 0.838. Quant à la scalabilité, la meilleure valeur que nous pouvons avoir pour le template est égale à (1 + 1) /2= 1. Ces résultats correspondent parfaitement aux scores de sécurité et de scalabilité proposés par notre solution pour le template, qui sont respectivement égaux à 0.838 et 1. Notre solution propose le plan SQL "Amazon - db.r3.8xlarge (Provisioned IOPS - monoAZ) " qui coûte 3167 $ et le plan de file de messages "SQS Standard" qui coûte 4 $.

— Nous remarquons que cette solution offre pour le template une valeur de performance élevée égale à 0.517 (la meilleure valeur de performance par rapport à la contrainte de prix) et une valeur de fiabilité élevée égale à 0.943 (0.943 contre 0.973 par rapport à la valeur maximale). Cependant, nous remarquons une diminution significative

		Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	prix	Score Instance
Besoin 1		35 pts	10 pts	35 pts	10 pts	10 pts		
Solution de Base	ServiceSQL : Amazon - db.r4.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.401	1.0	0.926	1793 $	
	FileAttente1 : Heroku CloudAMQP - RabbitMQ - Tought Tiger	1.0	0.902	0.390	0.5	0.737	19 $	
	NFR Instance template	1.0	0.929	0.397	0.75	0.831	1812 $	0.739
Solution alternative	ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.404	1.0	0.926	1769 $	
	FileAttente1 : SQS Standard	1.0	0.988	0.155	1.0	0.737	4 $	
	NFR Instance template	1.0	0.906	0.374	0.75	0.831	1873 $	0.729
Besoins 2		35 pts	35pts	10 pts	10 pts	10 pts		
Solution de Base	ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.404	1.0	0.926	1769 $	
	FileAttente1 : SQS Standard	1.0	0.988	0.155	1.0	0.737	4 $	
	NFR instance template	1.0	0.973	0.279	1.0	0.832	1773 $	0.901
Solution alternative	ServiceSQL : Amazon - db.r4.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.401	1.0	0.926	1793 $	
	FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $	
	NFR instance template	1.0	0.973	0.275	1.0	0.832	1798 $	0.901
Besoins 3		10 pts	10 pts	10 pts	35 pts	35pts		
Solution de Base	ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.404	1.0	0.926	1769 $	
	FileAttente1 : SQS Standard	1.0	0.988	0.155	1.0	0.737	4 $	
	NFR instance template	1.0	0.973	0.279	1.0	0.832	1773 $	0.866
Solution alternative	ServiceSQL : Amazon - db.r4.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.401	1.0	0.926	1793 $	
	FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $	
	NFR instance template	1.0	0.973	0.275	1.0	0.832	1798 $	0.866

TABLE 6.12 – Résultats de la solution de base et ses alternatives pour les besoins des trois templates pour un prix de 1980$

		Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	prix	Score template
Besoin 1		35 pts	10 pts	35 pts	10 pts	10 pts		
Solution de Base	ServiceSQL : Amazon - db.r4.16xlarge (General Purpose - multiAZ)	1.0	0.957	0.990	1.0	0.926	6191 $	
	FileAttente : cloudamqp -RabbitMQ - Loud Lion - 2 nodes	1.0	0.853	0.64	0.5	0.737	3499 $	
	NFR Instance template	1.0	0.905	0.815	0.75	0.832	9690 $	0.883
Solution alternative	ServiceSQL : Amazon - db.r4.16xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.993	1.0	0.926	12571 $	
	FileAttente : Heroku CloudAMQP -RabbitMQ - Heavy Hippo	1.0	0.902	0.515	0.5	0.737	1999 $	
	NFR Instance template	1.0	0.929	0.754	0.75	0.832	14570 $	0.865
Besoins 2		35 pts	35pts	10 pts	10 pts	10 pts		
Solution de Base	ServiceSQL : Amazon - db.r4.16xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.993	1.0	0.926	12571 $	
	- FileAttente : SQS Standard	1.0	0.988	0.155	1.0	0.737	4 $	
	NFR instance template	1.0	0.973	0.574	1.0	0.832	12575 $	0.931
Solution alternative	ServiceSQL : Amazon - db.m4.16xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.963	1.0	0.926	9265 $	
	FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $	
	NFR instance template	1.0	0.973	0.556	1.0	0.832	9270 $	0.929
Besoins 3		10 pts	10 pts	10 pts	35 pts	35pts		
Solution de Base	ServiceSQL : Amazon - db.r4.16xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.993	1.0	0.926	12571 $	
	-FileAttente : SQS Standard	1.0	0.988	0.155	1.0	0.737	4 $	
	NFR instance template	1.0	0.973	0.574	1.0	0.832	12575 $	0.896
Solution alternative	ServiceSQL : Amazon - db.m4.16xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.963	1.0	0.926	9265 $	
	FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $	
	NFR instance template	1.0	0.973	0.556	1.0	0.832	9270 $	0.894

TABLE 6.14 – Résultats de la solution de base et ses alternatives pour les besoins des trois templates pour un prix illimité

du score de la disponibilité du template de l'exigence 3. Nous avons un score de disponibilité de 0.740 pour le template du besoin 3 contre une valeur de 1 pour le template de l'exigence 1 et de l'exigence 2.

— Une solution alternative est de remplacer dans la solution de base le plan "SQS Standard" qui coûte 4$ par le plan "SQS FIFO" qui coûte 5$. Ces deux plans ont exactement la même configuration sur tous les NFR à l'exception de la performance où nous baissons légèrement le score. Nous pouvons considérer cette petite différence de score comme négligeable (0.155 pour le score de la performance du plan "SQS Standard" contre "0.150" pour score de la performance du service SQS FIFO). En effet, ces deux plans sont similaires dans leur configuration, la différence réside dans le scénario de leur utilisation. Dans l'hypothèse où nous aurions besoin d'utiliser une file de messages FIFO, nous devrons choisir le plan "SQS FIFO". Sinon, nous pouvons utiliser les deux plans.

— Nous pouvons considérer également que la solution obtenue pour "template besoin 2 et alternative template besoin 1" comme une solution alternative pour le besoin 3 du template. Pour cette solution alternative nous avons des valeurs élevées pour la sécurité (0.832 contre 0.838 pour la solution de base) et une valeur de scalabilité maximale égale à 1 qui est identique à celle de la solution de base. Nous remarquons que cette solution alternative est meilleure que la solution de base en score de disponibilité (score de disponibilité égal à 1 pour la solution alternative contre un score de 0.740 pour la solution de base). Cependant, la solution alternative est moins bonne que la solution de base en score de performance (score de performance égal à 0,354 pour la solution alternative contre un score de 0,515 pour la solution de base).

6.5.1.2 Sélection de templates pour un coût de 1980 $

Comme pour les expériences précédentes, nous avons utilisé notre méthodologie pour classer les instances de template en solutions de base et alternatives en tenant compte des différentes exigences non fonctionnelles présentées dans la Table 6.8. La configuration des plans de services obtenus pour ses différents templates est présentée dans la Table 6.12.

Besoin 1 Comme mentionné dans la Table 6.8 le besoin 1 du template exige, d'une part, des performances (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part,

une sécurité (10 pts), une fiabilité (10 pts) et une scalabilité (10 pts) moins importantes pour une contrainte de prix de 1980 $.

— *Pour le service de base de données SQL :* Comme le montre la Table 6.12, pour répondre à l'exigence 1 pour une contrainte de prix de 1980 $, la meilleure valeur de performance que nous pouvons avoir pour le service SQL est égale 0.404. De plus, la meilleure valeur de disponibilité qu'un service de même type peut offrir est égale à 1. Ces valeurs correspondent au plan de service" Amazon - db.r4.2xlarge(Provisioned IOPS -multiAZ)" qui coûte 1793 $. Ce plan de service est bien proposé par notre solution.

— *Pour le service de file de messages :* Comme le montre la Table 6.12, la meilleure valeur de performance qu'un sevice de file de messages peut fournir est égale 0.39. De plus la meilleure valeur de disponibilité que nous pouvons avoir pour ce même type de service est égale à 1. Notre solution propose le plan de service " "Heroku-CloudAMQP - RabbitMQ -Tought Tiger" ayant des valeurs de performance et de disponibilité respectivement égales à 0.39 et 1 pour un coût de 19 $.

— En termes de performance, la meilleure valeur que nous pouvons avoir pour le template est égale à $(0.404 + 0.39)\ /2 = 0.397$. Quant à la disponibilité, la meilleure valeur que nous pouvons avoir pour les services composant le template est égale à 1. Ces résultats correspondent parfaitement aux scores de performances et de disponibilité proposée par notre solution pour le template, qui sont respectivement égaux à 0.397 et 1.

En effet, notre solution propose le plan SQL "Amazon - db.r4.2xlarge (Provisioned IOPS -multiAZ) " qui coûte 1793 $ et le plan de file de messages " HerokuCloudAMQP - RabbitMQ -Tought Tiger " qui coûte 19 $. Nous remarquons également que cette solution offre pour le template une valeur de fiabilité élevée (égal à 0.929 contre 0.973 par rapport à la valeur maximale de la fiabilité du template) et une valeur de scalabilité moyenne (égal à 0.75 contre 1 par rapport à la valeur maximale de scalabilité du template).

— Une solution alternative est de remplacer dans la solution de base le plan " Heroku-CloudAMQP - RabbitMQ -Tought Tiger " du service de file d'attente qui coûte 19$ par le plan "RabbitMQ -Power Panda - 3 nodes -multiAZ" qui coûte 499$ et de remplacer dans la solution de base le plan " Amazon db.r3.2xlarge(Provisioned IOPS -

multiAZ) " du service SQL qui coûte 1769 $ par le plan " Amazon -db.m4.2xlarge(Provisioned IOPS -multiAZ) " qui coûte 1374 $. On constate qu'avec cette solution nous baissons légèrement le score de performance (un score de performance égal à 0.374 pour la solution alternative contre un score de performance égal à 0.397 pour la solution de base) et en score de fiabilité (un score de fiabilité égal à 0.906 pour la solution alternative contre un score de fiabilité égal à 0.929 pour la solution de base) pour un coût de 1873 $ pour la solution alternative contre un coût de 1812 $ pour la solution de base.

Besoin 2 Comme mentionné dans la Table 6.8 le besoins 2 du template exige, d'une part, une fiabilité (35 pts) et une disponibilité plus importantes (35 pts) et, d'autre part, une sécurité (10 pts), une performance (10 pts) et une scalabilité (10 pts) moins importantes, le tout pour une contrainte de prix de 1980 $.

— *Pour le service de base de données SQL :* Comme le montre la Table 6.12, les meilleures valeurs de disponibilité et fiabilité que nous pouvons avoir pour le plan de service SQL sont égales respectivement à 1 et 0,957. Notre solution propose le plan SQL " ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ) " qui répond le mieux au besoin 2 avec une valeur de disponibilité égale à 1 et une valeur de fiabilité égale à 0.957. Ce plan de service coûte 1769 $.

— *Pour le service de file de messages :* Comme le montre la Table 6.12, la meilleure valeur de disponibilité pour le service de file de messages est également égale à 1. De plus, la meilleure valeur de fiabilité que nous pouvons avoir pour ce même type de service est égale à 0,988. Notre solution propose le plan de file de messages "SQS Standard" qui coûte 4 $ ayant des valeurs de disponibilité et de fiabilité respectivement égales aux meilleures valeurs 1 et 0,988 pour les services de file de messages.

— En termes de disponibilité, la meilleure valeur que nous pouvons avoir pour le template est égale à 1. Quant à la fiabilité, la meilleure valeur que nous pouvons avoir pour le template est égale à (0.957 + 0.988) /2= 0.973. Ces résultats correspondent parfaitement aux scores de disponibilité et de fiabilité proposés par notre solution pour le template, qui sont respectivement égaux à 1 et 0.973. En effet, notre solution propose le plan SQL " ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS -

multiAZ) " qui coûte 1769 $ et le plan de file de messages "SQS Standard" qui coûte 4 $. Nous remarquons que cette solution offre pour le template une valeur de scalabilité élevée égale à 1 et une valeur de sécurité élevée égale à 0.832 (la valeur de sécurité maximale). Cependant, nous remarquons une diminution du score de la performance du template de l'exigence 2. Nous avons un score de 0.275 pour le template du besoin 2 contre une valeur de 0.397 pour le template de l'exigence 1.

— Une solution alternative consiste à remplacer dans la solution de base le plan "SQS Standard" qui coûte 4 $ par le plan "SQS FIFO" qui coûte 5 $. Ces deux plans ont exactement la même configuration sur tous les NFR sauf pour les performances où nous baissons légèrement le score (cette différence dans le score de performance peut être considérée comme négligeable). Dans cette solution alternative nous remplaçons également dans la solution de base le plan "Amazon - db. r3. 2xlarge (Provisioned IOPS - multiAZ)" qui coûte 1769 $ par le plan "Amazon - db.r4.2xlarge (Provisioned IOPS - multiAZ)" qui coûte 1793 $. Ces deux plans ont exactement la même configuration sur tous les NFR sauf pour les performances où nous baissons légèrement le score (cette différence de score peut être considérée comme négligeable). Cette solution alternative a exactement les mêmes valeurs de score NFR que la solution de base. Par conséquent, elle constitue une bonne solution pour le besoin du template 2.

Besoin 3 Comme indiqué dans la Table 6.8 le besoin 3 du template exige, d'une part, une sécurité (35 pts) et une scalabilité (35 pts) plus importantes et, d'autre part, une fiabilité (10 pts), une performance (10 pts) et une disponibilité (10 pts) moins importantes, le tout pour une contrainte de prix de 1980 $.

— *Pour le service de base de données SQL :* comme le montre la Table 6.12, les meilleures valeurs de sécurité et scalabilité pour un service SQL sont égales respectivement à 0.939 et 1. Notre solution propose le plan de service SQL " Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ " qui coûte 1769 $ ayant des valeurs de sécurité et de scalabilité respectivement égales aux valeurs maximales de 0.926 et 1. Ce plan est le mieux adapté pour répondre aux exigences du besoin 3.

— *Pour le service de file de messages :* Comme le montre la Table 6.12, la meilleure valeur de de sécurité que nous pouvons avoir pour le plan de service file de messages

est égale à 0.737. Quant à la scalabilité, la meilleure valeur que ce même type de service peut fournir est égale à 1. Notre solution propose le plan de service le plan de file de messages "SQS Standard" qui coûte 4 $ ayant des valeurs de sécurité et de scalabilité respectivement égales aux valeurs maximales de 0.737 et 1.

— En termes de sécurité, la meilleure valeur que nous pouvons avoir pour le template est égale à 0.838. Quant à la scalabilité, la meilleure valeur que nous pouvons avoir pour le template est égale à (1 + 1) /2= 1. Ces résultats correspondent parfaitement aux scores de sécurité et de scalabilité proposés par notre solution pour le template, qui sont respectivement égaux à 0.838 et 1. En effet, notre solution propose le plan SQL " Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ " qui coûte 1769 $ et le plan de file de messages "SQS Standard" qui coûte 4 $. Nous remarquons que cette solution offre pour le template une valeur de performance faible (0.279 contre 0.397 par rapport à la valeur maximale selon la contrainte de prix), une valeur de fiabilité élevée égale à 0.973 (la meilleure valeur de fiabilité par rapport à la contrainte de prix) et une valeur de disponibilité élevée égale à 1.

— Une alternative ayant exactement la même configuration que la solution de base, composée du service de file de messages "SQS FIFO" qui coûte 5 $ et du Service SQL" Amazon - db.r4.2xlarge (Provisioned IOPS - multiAZ)" qui coûte 1793 $ est présentée dans la Table 6.12.

6.5.1.3 Sélection de templates pour un coût illimité

Besoin 1 Comme mentionné dans la Table 6.8 le besoin 1 du template exige, d'une part, des performances (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une fiabilité (10 pts) et une scalabilité (10 pts) moins importantes pour un prix illimité.

— *Pour le service de base de données SQL :* Comme le montre la Table 6.13, la meilleure valeur de performance que nous pouvons avoir pour le service SQL est égale à 0.993. Cette valeur est fournie par le plan de service SQL "Amazon - db.r4.16xlarge (Provisioned IOPS - multiAZ)" qui coûte 12571 $. Notre solution propose ce dernier plan SQL ayant des valeurs de performances et de disponibilité respectivement égales aux meilleures valeurs 0.993 et 1.

— *Pour le service de file de messages :* la meilleure valeur de performance que nous

pouvons avoir pour le service de file de messages est égal à 0.89. Cette valeur est fournie par le plan de service "cloudamqp- RabbitMQ - Loud Lion - 1 node" qui coûte 3499 $. Cependant, le score de disponibilité pour ce dernier plan de file d'attente est nul. La deuxième meilleure valeur de performance pour le service de file d'attente est égale à 0.64 pour le plan de service 'cloudamqp - RabbitMQ - Loud Lion - 2 nodes'. Notre solution propose ce dernier plan de file de messages qui coûte 3499 $ ayant des valeurs de performance et de disponibilité respectivement égales aux valeurs de 0.64 et 1.

— En termes de performance, la meilleure valeur que nous pouvons avoir avec une disponibilité élevée pour les plans composant les template sont égaux à (0.993 + 0.64) /2 = 0.816 pour la performance et 1 (1+1/2) pour la disponibilité. Ces résultats correspondent parfaitement aux scores de performances et de disponibilité proposés par notre solution pour le template, qui sont respectivement égaux à 0.993 et 1. En effet, notre solution propose le plan SQL "Amazon - db.r4.16xlarge (Provisioned IOPS - multiAZ)" qui coûte 12571 $ et le plan de file de messages "cloudamqp - RabbitMQ - Loud Lion - 2 nodes" qui coûte 3499 $. Nous remarquons également que cette solution offre pour le template une valeur de fiabilité élevée (égal à 0.905 contre 0.973 par rapport à la valeur maximale de la fiabilité du template) et une scalabilité moyenne (égal à 0.75 contre 1 par rapport à la valeur maximale de scalabilité du template).

— Une solution alternative est de remplacer dans la solution de base le plan " cloudamqp -RabbitMQ - Loud Lion -2 nodes " du service de file d'attente qui coûte 3499 $ par le plan "Heroku CloudAMQP -RabbitMQ - Heavy Hippo " qui coûte 1999 $ et de remplacer dans la solution de base le plan " Amazon -db.r4.16xlarge (General Purpose -multiAZ)" du service SQL qui coûte 6191 $ par le plan " Amazon -db.r4.16xlarge (Provisioned IOPS -multiAZ) " qui coûte 12571 $. On constate qu'avec cette solution on obtient moins en score de performance (un score de performance égal à 0.754 pour la solution alternative contre un score de performance égal à 0.815 pour la solution de base) et on obtient un peu plus en score de fiabilité (un score de fiabilité de 0.929 pour la solution alternative contre un score de fiabilité de 0.905 pour la solution de base) pour un coût de 14570 $ pour la solution alternative contre un coût de 9690 $ pour la solution de base

Besoin 2 Comme mentionné dans la Table 6.8 le deuxième besoin du template requiert, d'une part, une fiabilité (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une performance (10 pts) et une scalabilité (10 pts) moins importantes, le tout pour un coût illimité.

- *Pour le service de base de données SQL* : Comme le montre la Table 6.13, le plan de service SQL qui répond le mieux au besoin 2 est "ServiceSQL : " Amazon - db.r4.16xlarge (Provisioned IOPS - multiAZ " avec un score de disponibilité maximale égale à 1 et un score de fiabilité élevé égale à 0,957 pour un prix de 12571 $.
- *Pour le service de file de messages* : Comme le montre, la Table 6.13, le meilleur plan de service correspond au "SQS standard" avec un score de disponibilité maximale égale à 1 et un score de fiabilité élevé égale à 0,988 pour un prix de 4$.
- En termes de fiabilité, la meilleure valeur que nous pouvons avoir pour le template est égale à (0.957 + 0.988) /2= 0.973. Ces résultats correspondent parfaitement aux scores de disponibilité et de fiabilité proposés par notre solution pour le template, qui sont respectivement égaux à 1 et 0.973. En effet, notre solution propose le plan SQL "ServiceSQL : Amazon - db.r4.16xlarge (Provisioned IOPS - multiAZ " qui coûte 12571 $ pour les services SQL. Elle propose le plan de file de messages "SQS Standard" qui coûte 4 $ pour les services de file de messages. Nous remarquons que cette solution offre, également, pour le template une valeur de scalabilité élevée égale à 1 et une valeur de sécurité élevée égale à 0.832. Cette valeur de sécurité varie légèrement par rapport à la valeur maximale (0.838). Nous remarquons notamment que pour un prix illimité nous avons un bon score de performance égal à 0.574 par rapport aux prix précédents du même besoin (un score de 0.279 pour un prix de 1980 $ et un score de 0.354 pour un prix de 3680 $ illustrés respectivement dans les Tables 6.12 et 6.15)
- Une solution alternative est de remplacer dans la solution de base le plan "SQS Standard" qui coûte 4$ par le plan "SQS FIFO" qui coûte 5 $. Ces deux plans ont exactement la même configuration sur tous les NFR à l'exception de la performance où nous baissons légèrement le score. Nous pouvons considérer cette petite différence de score comme négligeable (score de 0.155 pour la performance du plan "SQS Standard" contre un score de "0.150" pour la performance du service SQS FIFO). En

effet, ces deux plans sont similaires dans leur configuration, la différence réside dans le scénario de leur utilisation. Dans l'hypothèse où nous aurions besoin d'utiliser une file de messages FIFO, nous devrons choisir le plan "SQS FIFO". Sinon, nous pouvons utiliser les deux plans

Besoin 3 Comme indiqué dans la Table 6.8 le besoin 3 du template exige, d'une part, une sécurité (35 pts) et une scalabilité (35 pts) plus importantes et, d'autre part, une fiabilité (10 pts), une performance (10 pts) et une disponibilité (10 pts) moins importantes, le tout pour un coût illimité.

— *Pour le service de base de données SQL :* comme le montre la Table 6.13, les meilleures valeurs de sécurité et scalabilité pour un service SQL sont égales respectivement à 0.939 et 1. Notre solution propose le plan de service SQL " Amazon -db.m4.16xlarge (Provisioned IOPS - multiAZ)" qui coûte 9270 $ ayant des valeurs de sécurité et de scalabilité respectivement égales aux valeurs de 0.926 et 1. Ce plan est le mieux adapté pour répondre aux exigences du besoin 3.

— *Pour le service de file de messages :* Comme le montre la Table 6.13, la meilleure valeur de de sécurité que nous pouvons avoir pour le plan de service file de messages est égale à 0.737. Quant à la scalabilité, la meilleure valeur que ce même type de service peut fournir est égale à 1. Notre solution propose le plan de service le plan de file de messages "SQS Standard" qui coûte 4 $ ayant des valeurs de sécurité et de scalabilité respectivement égales aux valeurs maximales de 0.737 et 1.

— En termes de sécurité, la meilleure valeur que nous pouvons avoir pour le template est égale à 0.838. Quant à la scalabilité, la meilleure valeur que nous pouvons avoir pour le template est égale à (1 + 1) /2= 1. Ces résultats correspondent parfaitement aux scores de sécurité et de scalabilité proposés par notre solution pour le template, qui sont respectivement égaux à 0.832 et 1. En effet, notre solution propose le plan SQL " Amazon -db.m4.16xlarge (Provisioned IOPS - multiAZ) " qui coûte 9270 $ et le plan de file de messages "SQS Standard" qui coûte 4 $. Nous remarquons notamment que pour un prix illimité on obtient un bon score de performance égal à 0.574 et un excellent score de disponibilité égal à 1 par rapport aux prix précédents du même besoin (0.279 de performance et 1 de disponibilité pour un prix de 1980 $; 0.517 de performance et 0.740 de disponibilité pour un prix de 3680 $ comme

illustrés respectivement dans les Tables 6.12 et 6.15)

— Une alternative ayant exactement la même configuration que la solution de base, composée du service de file de messages "SQS FIFO" qui coûte 5 $ et du Service SQL " Amazon -db.m4.16xlarge (Provisioned IOPS - multiAZ) qui coûte 9270 $ est présentée dans la Table 6.13.

Dans cette section, nous avons fait varier les exigences des templates en fixant la contrainte de coût pour chaque scénario. De cette façon, nous avons pu justifier, pour chaque contrainte de prix, les scores NFR obtenus pour chaque exigence et vérifier que ces scores correspondent aux exigences spécifiées par les architectes selon la contrainte de prix de chaque template.

6.5.2 Validation de la variation des scores NFR en fonction des besoins et des prix

Dans la section précédente, nous avons défini des scénarios dans lesquels nous avons comparé différentes exigences de template en fixant le coût de chaque template. De cette façon, nous avons visualisé la variation des scores NFR en fonction du prix pour chaque exigence. Nous avons ainsi pu vérifier que nous avons obtenu des solutions adaptées aux exigences spécifiées pour chaque prix à partir des configurations des plans de services. Dans cette section, nous définissons des scénarios dans lesquels nous fixons les exigences des templates et faisons varier les prix. De cette façon, nous pouvons examiner la variation des valeurs des scores NFR en fonction du prix pour les mêmes exigences de template. Nous pouvons ainsi justifier la variation des scores NFR en fonction du prix.

Besoin 1 Comme mentionné dans la Table 6.8 le besoin 1 du template exige, d'une part, des performances (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une fiabilité (10 pts) et une scalabilité (10 pts) moins importantes pour un prix de 3960 $. Comme le montre la Table 6.16, pour les trois scénarios avec les prix 3960 $, 1980 $ et illimité, les valeurs NFR sont toutes les mêmes sauf pour la performance où nous avons une valeur égale à 0.481 pour un prix de 3960 $, une valeur de 0.397 pour un prix de 1980 $ et 0.81 pour un prix illimité.

Nous constatons que nous obtenons les meilleures valeurs en termes de disponibilité, comme l'exige ce besoin, pour les trois prix. En particulier, nous remarquons que chaque

		Instance template						
		Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	Score Globale	Prix
Besoin 1		35	10	35	10	10		
Prix 1=1980	- ServiceSQL : Amazon - db.r4.2xlarge (Provisioned IOPS - multiAZ) - FileAttente1 : Heroku CloudAMQP - RabbitMQ - Tought Tiger	1.0	0.929	0.397	0.75	0.831	0.927	1812 $
Prix 2= 3960	- ServiceSQL :Amazon-db.r3.4xlarge (Provisioned IOPS -multiAZ) -cloudamqp- RabbitMQ - Power Panda - 3 nodes -multiAZ	1.0	0.906	0.481	0.75	0.832	0.956	3790$
Prix 3=illimité	- ServiceSQL : Amazon - db.r4.16xlarge (General Purpose - multiAZ) - FileAttente : cloudamqp -RabbitMQ- Loud Lion - 2 nodes "	1.0	0.905	0.815	0.75	0.832	0.901	9690 $
Besoin 2		35	35	10	10	10		
Prix 1=1980	- ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ) - FileAttente1 : SQS Standard	1.0	0.973	0.279	1.0	0.832	0.950	1773 $
Prix 2=3960	- ServiceSQL : Amazon - db.r4.4xlarge (Provisioned IOPS -multiAZ) - FileAttente1 :SQS Standard	1.0	0.973	0.354	1.0	0.832	0.954	3343$
Prix 3=illimité	- ServiceSQL : Amazon - db.r4.16xlarge (Provisioned IOPS - multiAZ) - FileAttente : SQS Standard	1.0	0.973	0.574	1.0	0.832	0.965	12575 $
Besoin 3		10	10	10	35	35		
Prix 1=1980	-ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ) -FileAttente1 : SQS Standard	1.0	0.973	0.279	1.0	0.832	0.950	1773 $
Prix 2= 3989	- ServiceSQL : Amazon - db.r3.8xlarge (Provisioned IOPS - monoAZ) - FileAttente1 :SQS Standard	0.740	0.943	0.517	1.0	0.838	0.798	3172 $
Prix 3=illimité	- ServiceSQL : Amazon - db.r4.16xlarge (Provisioned IOPS - multiAZ) - FileAttente : SQS Standard	1.0	0.973	0.556	1.0	0.832	0.831	9270 $

TABLE 6.16 – Résultats de la solution de base pour chaque besoin des trois templates en faisant varier les prix

fois que nous augmentons le prix, nous obtenons plus de performance.

Besoin 2 Le deuxième besoin du template requiert, d'une part, une fiabilité (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une performance (10 pts) et une scalabilité (10 pts) moins importantes. Comme le montre la Table 6.16, pour les trois scénarios avec les prix 3960 $, 1980 $ et illimité, les valeurs NFR sont toutes les mêmes sauf pour la performance où, nous avons une valeur égale 0.354 pour un prix de 3960 $, une valeur de 0.279 pour 1980 $ et 0.574 pour un prix illimité.

Nous constatons que nous obtenons les meilleures valeurs en termes de disponibilité et de fiabilité, comme l'exige ce besoin, pour les trois prix. En particulier, nous remarquons que chaque fois que nous augmentons le prix, nous obtenons plus de performance.

Besoin 3 Comme indiqué dans la Table 6.8 le besoin 3 du template exige, d'une part, une sécurité (35 pts) et une scalabilité (35 pts) plus importantes et, d'autre part, une fiabilité (10 pts), une performance (10 pts) et une disponibilité (10 pts) moins importantes. Comme le montre la Table 6.16, nous remarquons que nous obtenons les meilleures valeurs en termes de scalabilité et de sécurité, comme l'exige ce besoin, pour les trois prix. En particulier, nous remarquons que chaque fois que nous augmentons le prix, nous obtenons plus de performance. De plus, nous remarquons que le score de performance peut augmenter pour le même template mais nous diminuons sur le score de disponibilité.

Par exemple pour le deuxième prix 3960 $, nous remarquons que par rapport aux exigences précédentes, nous augmentons le score de performance, qui est égal à 0,517 (contre 0,354 pour les exigences précédentes). Cependant, nous diminuons le score de disponibilité qui est égale à 0.740 (par rapport à 1 précédemment).

Conclusion Nous avons présenté dans ce travail une première validation de notre méthodologie pour guider les architectes dans la sélection des templates en fonction d'un ensemble d'exigences et de contraintes. Ainsi, nous avons présenté dans la section précédente, 3 scénarios principaux pour comparer le même template avec les mêmes services sous différentes exigences non-fonctionnelles et différentes contraintes de coûts. Un scénario consistait à fixer le coût global du template et à modifier les exigences à chaque fois, ainsi au total, nous avions 9 scénarios. Pour chaque scénario, nous avions fait varier les exigences des templates en fixant la contrainte de coût. De cette manière, nous étions en

mesure de justifier, pour chaque contrainte de prix, les scores NFR obtenus pour chaque exigence et de vérifier que ces scores correspondaient aux exigences spécifiées par les architectes. Ensuite, dans cette section, nous avons défini des scénarios dans lesquels nous avons comparé différentes exigences de template en fixant le coût de chaque template. De cette façon, nous avons visualisé la variation des scores NFR en fonction du prix pour chaque exigence. Nous pouvons constater que les templates ont la même sécurité, la même scalabilité et presque la même fiabilité indépendamment des contraintes de coût, et ce en choisissant les mêmes fournisseurs de services. Cependant, lorsque nous changeons le prix, le score de disponibilité et le score de performance changent pour les mêmes fournisseurs. Cette constatation peut s'expliquer par le fait que pour le même fournisseur, les valeurs de sécurité et de scalabilité des plans de service restent les mêmes quel que soit le prix. Cependant, ce constat diffère pour les deux NFR, dont la performance et la disponibilité, qui semblent être celles qui déterminent et influencent le prix des plans de services. Le fournisseur Amazon en est un exemple. Chaque fois que la valeur de la performance augmente, le prix augmente également. De plus, si le client veut plus de disponibilité, c'est-à-dire bénéficier de la réplication sur une deuxième instance, celui-ci acquitte le double du prix de l'instance réservée (2 instances).

Nous pensons qu'une validation auprès d'un plus grand nombre de fournisseurs de services est nécessaire pour confirmer nos affirmations.

6.5.3 Validation des choix des instances de templates pour des architectures techniques

Pour valider notre approche, nous avons utilisé 3 scénarios principaux pour comparer les instances d'une même architecture. Cette architecture est composée trois fois du même template, chacun d'entre eux ayant les différentes exigences non fonctionnelles présentées dans la table 6.8. Un scénario consiste à définir le coût global de l'architecture et à générer ultérieurement un ensemble d'instanciations de template en fonction des exigences. À son tour, chaque template de services cloud est composé de deux types de services : Les services de base de données SQL et les services cloud file de messages. Ces services ont les mêmes exigences que celles spécifiées précédemment dans la Table 6.6 et la Table 6.7.

Afin de vérifier que nous obtenons des instanciations de templates adaptées aux besoins spécifiés pour les trois templates composant l'architecture, nous les comparons aux tem-

plates de référence. Un template de référence correspond au template ayant les meilleurs scores NFR pour un prix donné. La Table 6.17 représente les templates de référence pour les différents prix considérés.

Contrainte de prix pour l'architecture	le meilleur score de disponibilité du template	le meilleur score de fiabilité du template	le meilleur score de performance du template	le meilleur score de scalabilité du template	le meilleur score de sécurité du template
Illimité	1.0	0.973	- 0.941 avec un score de disponibilité égal à 0.5 - 0.816 avec un score de disponibilité égal à 1	1.0	0.838
11000 $	1.0	0.973	- 0.517 avec un score de disponibilité égal à 0.740 - 0.481 avec un score de disponibilité égal à 1	1.0	0.838
6000 $	1.0	0.973	0.397 avec un score de disponibilité égal à 1	1.0	0.838

TABLE 6.17 – Les meilleurs scores NFR au niveau des différents plans en fonction de la contrainte de prix.

Dans cette section, pour chaque contrainte de coût, nous présentons uniquement les solutions de base obtenues pour l'architecture considérée sans mentionner leurs alternatives.

6.5.3.1 Sélection d'une architecture de templates pour un coût de 6000 $

Nous avons appliqué notre méthodologie pour classer les instances de templates composant l'architecture en solutions de base et alternatives, en tenant compte des exigences non fonctionnelles des templates (Tableau 6.8) et de la contrainte de coût. Cette dernière consiste à ce que le coût de l'ensemble des services composant l'architecture ne doive pas dépasser une somme de 6000 $. La configuration des plans de services obtenus pour ces différents templates est présentée dans la Table 6.18.

Pour chaque besoin, nous discutons et évaluons nos résultats en les comparant aux templates de référence présentant les meilleures valeurs de NFR selon les contraintes de prix présentées dans la table 6.17.

Besoin 1 Comme mentionné dans la Table 6.8 le besoin 1 du template requiert, d'une part, des performances (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une fiabilité (10 pts) et une scalabilité (10 pts) moins importantes.

Comme le montre la table 6.18, nous obtenons un score de disponibilité maximale égal à 1 et un score de performance élevé égal à 0,397. Le score de performance pour ce prix, représente le meilleur score de performance qui s'accompagne également d'un score de disponibilité élevé. En ce qui concerne les scores des autres NFR et indépendamment de la contrainte de prix, nous obtenons les valeurs suivantes : (i) Un score de scalabilité de 0,75 , qui est inférieur au meilleur score qui est égal à 1 ; (ii) Un score de fiabilité égal à 0.929, ce qui est légèrement inférieur au meilleur score, qui est égal à 0,973 pour le meilleur score de scalabilité pour les templates ; (iii) Un score de sécurité élevé égal à 0,831, ce qui est légèrement inférieur au meilleur score de sécurité 0,838 pour les templates. Le coût de cette instance de modèle est égal à 1812 $.

Besoin 2 Comme mentionné dans la Table 6.8 le besoin 2 du template requiert, d'une part, une fiabilité (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une performance (10 pts) et une scalabilité (10 pts) moins importantes.

Comme le montre la Table 6.18, nous obtenons un score de disponibilité maximal de 1 et un excellent score de fiabilité de 0,973. Ce score de fiabilité correspond au meilleur score pour cette NFR obtenu pour les templates quelle que soit la contrainte de prix. En ce qui concerne les scores des autres NFR, nous obtenons un score de scalabilité maximale égal à 1, un score de sécurité élevé égal à 0,831. Le score de sécurité est légèrement inférieur au meilleur score de cette NFR qui est égal à 0,838 pour les templates quelle que soit la contrainte de prix. Alors que pour la performance, nous obtenons un faible score égal à 0.279, sachant que le meilleur score de performance que nous pouvons obtenir pour cette contrainte de prix est égal à 0.397. Le coût de cette instance de modèle est égal à 1773 $.

Besoin 3 Comme mentionné dans la Table 6.8 le besoin 3 du template requiert, d'une part, une sécurité (35 pts) et une scalabilité (35 pts) plus importantes et, d'autre part, une fiabilité (10 pts), une performance (10 pts) et une disponibilité (10 pts) moins importantes.

Comme le montre la Table 6.18, nous obtenons un score de scalabilité maximale égal à 1 et un score de sécurité élevée égal à 0.831, sachant que le meilleur score de sécurité

			Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	Prix template	Score template	Prix Architecture	Score Architecture
Solution de base	Besoin 1		35 pts	10 pts	35 pts	10 pts	10 pts			5358 $	0.835
	template 1	ServiceSQL : Amazon - db.r4.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.401	1.0	0.926	1793 $			
		FileAttente1 : Heroku CloudAMQP - RabbitMQ - Tought Tiger	1.0	0.902	0.390	0.5	0.737	19 $			
		NFR template 1	1.0	0.929	0.397	0.75	0.831	1812 $	0.739		
	Besoin 2		35 pts	35pts	10 pts	10 pts	10 pts				
	template 2	ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.404	1.0	0.926	1769 $			
		FileAttente1 : SQS Standard	1.0	0.988	0.155	1.0	0.737	4 $			
		NFR template 2	1.0	0.973	0.279	1.0	0.831	1773 $	0.901		
	Besoin 3		10 pts	10 pts	10 pts	35 pts	35pts				
	template 3	ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.404	1.0	0.926	1769 $			
		FileAttente1 : SQS Standard	1.0	0.988	0.155	1.0	0.737	4 $			
		NFR templates	1.0	0.973	0.279	1.0	0.831	1773 $	0.866		
Solution de base	Besoin 1		35 pts	10 pts	35 pts	10 pts	10 pts			5469 $	0.832
	template 1	ServiceSQL : Amazon - db.r3.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.404	1.0	0.926	1769 $			
		FileAttente1 : SQS Standard	1.0	0.988	0.155	1.0	0.737	4 $			
		NFR template1	1.0	0.906	0.374	0.75	0.831	1873 $	0.729		
	Besoin 2		35 pts	35pts	10 pts	10 pts	10 pts				
	template 2	ServiceSQL : Amazon - db.r4.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.401	1.0	0.926	1793 $			
		FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $			
		NFR template 2	1.0	0.973	0.275	1.0	0.831	1798 $	0.901		
	Besoin 3		10 pts	10 pts	10 pts	35 pts	35pts				
	template 3	ServiceSQL : Amazon- db.r4.2xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.401	1.0	0.926	1793 $			
		FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $			
		NFR template 3	1.0	0.973	0.275	1.0	0.831	1798 $	0.866		

TABLE 6.19 – Sélection d'une architecture de templates pour un coût de 6000 $

est égal à 0,838 pour les templates indépendamment du prix. En ce qui concerne les scores des autres NFR, nous obtenons un score de performance faible égal à 0.279 , alors que le meilleur score de performance qui peut être obtenu pour cette contrainte de prix est égal à 0.397. De plus, nous obtenons un score de disponibilité maximale égal à 1 ainsi qu'un excellent score de fiabilité égal à 0.973. Le score de fiabilité correspond au meilleur score pour cette NFR obtenu pour les modèles indépendamment de la contrainte de prix. Le coût de cette instance de template est égal à 1773$.

6.5.3.2 Sélection d'une architecture de templates pour un coût de 11000 $

Comme pour les expériences précédentes, nous avons utilisé notre méthodologie pour classer les instances de template composant l'architecture en solutions de base et alternatives, en tenant compte des exigences non fonctionnelles des templates (Table 6.8) et de

la contrainte de coût. Cette dernière consiste à ce que le coût de l'ensemble des services composant l'architecture ne doive pas dépasser une somme 11000 $. La configuration des plans de services obtenus pour ses différents templates est présentée dans la Table 6.20.

Pour chaque besoin, nous discutons et évaluons nos résultats en les comparant aux templates de référence présentant les meilleures valeurs de NFR selon les contraintes de prix présentées dans la table 6.17.

Besoin 1 Comme mentionné dans la Table 6.8 le besoin 1 du template requiert, d'une part, des performances (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une fiabilité (10 pts) et une scalabilité (10 pts) moins importantes.

Comme le montre la Table 6.20, nous obtenons un score de disponibilité maximale de 1 et un score de performance élevée de 0,481 pour ce prix. Ces deux scores correspondent aux meilleurs scores de performance et de disponibilité. En ce qui concerne les scores des autres NFR, nous obtenons un score de scalabilité de 0,75 qui est inférieur au meilleur score "1" de ce NFR pour les templates, indépendamment de la contrainte de prix. De plus, nous obtenons un score de fiabilité de 0,906 qui est légèrement inférieur au meilleur score de 0,973. Nous obtenons également un score de sécurité élevé égal à 0,831, alors que le meilleur score de sécurité est égal à 0,838 pour les templates. Le coût de cette instance de modèle est égal à 3790 euros.

Besoin 2 Comme mentionné dans la Table 6.8 le besoin 1 du template requiert, d'une part, une fiabilité (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une performance (10 pts) et une scalabilité (10 pts) moins importantes.

Nous obtenons un score de disponibilité maximale égal à 1 et un excellent score de fiabilité égal à 0,973, qui est le meilleur score de fiabilité obtenu pour les templates, quelle que soit la contrainte de prix. En ce qui concerne les scores des autres NFR, nous obtenons un score de scalabilité maximal égal à 1, un score de sécurité élevé égal à 0,831, qui est légèrement inférieur au meilleur score de sécurité "0,838" pour les templates. En plus nous obtenons un score de performance modérément élevé. En fait, le meilleur score de performance est égal à 0,481 pour un template avec un score de disponibilité élevé. Le coût de cette instance de modèle est égal à 3343 $.

Besoin 3 Comme mentionné dans la Table 6.8 le besoin 3 du template requiert, d'une part, une sécurité (35 pts) et une scalabilité (35 pts) plus importantes et, d'autre part, une fiabilité (10 pts), une performance (10 pts) et une disponibilité (10 pts) moins importantes.

Comme le montre la Table 6.20, nous obtenons un score de scalabilité maximale égal à 1 et un score de sécurité élevé égal à 0.838, qui est le meilleur score de sécurité obtenu pour les templates indépendamment de la contrainte de prix. En ce qui concerne les scores des autres NFR, nous obtenons un score de performance très élevé égal à 0,517, qui est le meilleur score de performance obtenu à ce prix, mais associé à un score de disponibilité faible. De plus, nous avons obtenu un score de disponibilité de 0,74, ce qui est inférieur au meilleur score égal à 1 pour les templates. Nous obtenons également, un excellent score de fiabilité égal à 0.943, qui est légèrement inférieur au meilleur score de fiabilité "0.973" pour les templates quel que soit le prix. Le coût de cette instance de template est égal à 3171$.

Il est intéressant de noter, à partir de cette validation, que le score de performance peut être amélioré si le score de disponibilité est réduit. Par exemple pour le besoin 3, nous obtenons un score de 0,517 pour la performance et un score de 0,74 pour la disponibilité. De façon réciproque, le score de disponibilité peut être amélioré si le score de performance est réduit. Par exemple pour le besoin 1, nous obtenons un score de 0,481 pour la performance et un score de 1 pour la disponibilité.

6.5.3.3 Sélection d'une architecture de templates pour un coût illimité

Comme pour les expériences précédentes, nous avons utilisé notre méthodologie pour classer les instances de template composant l'architecture en solutions de base et alternatives, en tenant compte des exigences non fonctionnelles des templates (Table 6.8) et de la contrainte de coût. Cette dernière consiste à ce que le coût de l'ensemble des services composant l'architecture soit illimité. La configuration des plans de services obtenus pour ses différents templates est présentée dans la Table 6.22.

Besoin 1 Comme mentionné dans la Table 6.8 le besoin 1 du template requiert, d'une part, des performances (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une fiabilité (10 pts) et une scalabilité (10 pts) moins importantes.

			Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	Prix template	Score template	Prix Architecture	Score Architecture
Solution de base	Besoins 1		35 pts	10 pts	35 pts	10 pts	10 pts			10305 $	0.846
	template 1	ServiceSQL :Amazon-db.r3.4xlarge (Provisioned IOPS -multiAZ)	1.0	0.957	0.560	1.0	0.926	3291 $			
		cloudamqp- RabbitMQ- Power Panda - 3 nodes -multiAZ	1.0	0.855	0.402	0.5	0.737	499$			
		NFR template 1	1.0	0.906	0.481	0.75	0.831	3790$	0.767		
	Besoins 2		35 pts	35pts	10 pts	10 pts	10 pts				
	template 2	ServiceSQL : Amazon-db.r4.4xlarge (Provisioned IOPS -multiAZ)	1.0	0.957	0.553	1.0	0.926	3339$			
		FileAttente1 :SQS Standard	1.0	0.988	0.155	1.0	0.737	4$			
		NFR template 2	1.0	0.973	0.354	1.0	0.831	3343$	0.91		
	Besoins 3		10 pts	10 pts	10 pts	35 pts	35pts				
	template 3	ServiceSQL : Amazon - db.r3.8xlarge (Provisioned IOPS - monoAZ)	0.481	0.899	0.879	1.0	0.939	3167 $			
		FileAttente1 :SQS Standard∼	1.0	0.988	0.155	1.0	0.737	4$			
		NFR template 3	0.740	0.943	0.517	1.0	0.838	3171 $	0.863		
Solution de base	Besoins 1		35 pts	10 pts	35 pts	10 pts	10 pts			10477 $	0.849
	template 1	ServiceSQL :Amazon-db.r3.4xlarge (Provisioned IOPS -multiAZ)	1.0	0.957	0.560	1.0	0.926	3291 $			
		cloudamqp- RabbitMQ - Power Panda - 3 nodes -multiAZ	1.0	0.855	0.402	0.5	0.737	499$			
		NFR template 1	1.0	0.906	0.481	0.75	0.831	3790$	0.767		
	Besoins 2		35 pts	35pts	10 pts	10 pts	10 pts				
	template 2	ServiceSQL : Amazon - db.r4.4xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.553	1.0	0.926	3339$			
		FileAttente1 : SQS FIFO	1.0	0.988	0.150	1.0	0.737	5 $			
		NFR template 2	1.0	0.973	0.352	1.0	0.831	3344$	0.909		
	Besoins 3		10 pts	10 pts	10 pts	35 pts	35pts				
	template 3	ServiceSQL : Amazon - db.r4.4xlarge (Provisioned IOPS - multiAZ)	1.0	0.957	0.553	1.0	0.926	3339$			
		FileAttente1 :SQS Standard	1.0	0.988	0.155	1.0	0.737	4$			
		NFR template 3	1.0	0.972	0.354	1.0	0.831	3343 $	0.873		

TABLE 6.21 – Sélection d'une architecture de templates pour un coût de 11000 $

Comme le montre la Table 6.22, nous obtenons un score de disponibilité maximale égal à 1 et un score de performance égal à 0.816 qui correspond au meilleur score pour cette NFR . En ce qui concerne les scores des autres NFR, nous obtenons un score de scalabilité de 0,75 qui est inférieur au meilleur score "1" pour les templates. Nous obtenons également, un score de fiabilité de 0.906 qui est légèrement inférieur au meilleur score "0.973" pour les templates. En outre, nous obtenons un score de sécurité élevé, égal à 0.831, légèrement inférieur au meilleur score "0,838". Le coût de cette instance de template est égal à 16070 $.

Besoin 2 Comme mentionné dans la table 6.8 le besoin 1 du template requiert, d'une part, une fiabilité (35 pts) et une disponibilité (35 pts) plus importantes et, d'autre part, une sécurité (10 pts), une performance (10 pts) et une scalabilité (10 pts) moins importantes.

Comme le montre le tableau 6.22, nous obtenons un score de disponibilité maximale égal à 1 et un excellent score de fiabilité égal à 0,973, qui est le meilleur score de fiabilité obtenu pour les templates, indépendamment de la contrainte de prix. En ce qui concerne les scores des autres NFR, nous obtenons un score de scalabilité maximale égal à 1, un score de sécurité élevé égal à 0,831 et un score de performance moyennement élevé égal à 0,574. Pour un prix illimité, le meilleur score de performance associé à un score de disponibilité élevé est égal à 0,816. Le coût de cette instance de template est égal à 12575 $.

Besoin 3 Comme mentionné dans la table 6.8 le besoin 3 du template requiert, d'une part, une sécurité (35 pts) et une scalabilité (35 pts) plus importantes et, d'autre part, une fiabilité (10 pts), une performance (10 pts) et une disponibilité (10 pts) moins importantes.

Comme le montre la Table 6.22, nous obtenons un score de scalabilité maximale égal à 1 et un score de sécurité élevée égal à 0,831. En ce qui concerne les scores des autres NFR, nous obtenons un score de haute performance égal à 0.574, un score de disponibilité maximale égal à 1 et un excellent score de fiabilité égal à 0.973, qui est le meilleur score de fiabilité obtenu pour les modèles quelle que soit la contrainte du prix. Le coût de cette instance de modèle est égal à 12575$.

conclusion Dans cette section, nous avons présenté une validation de notre méthodologie pour sélectionner un ensemble de templates en fonction de leurs besoins et de la contrainte de coût imposée pour l'architecture. Ainsi, le coût total des plans de services composant

			Disponibilité	Fiabilité	Performance	Scalabilité	Sécurité	Prix template	Score template	Prix Architecture	Score Architecture
Solution de base	Besoins 1		35 pts	10 pts	35 pts	10 pts	10 pts			41220 $	0.903
	template 1	ServiceSQL : Amazon db.r4.16xlarge Provisioned IOPS (multiAZ) :12571.32	1.0	0.957	0.993	1.0	0.926	12571 $			
		FileAttente :cloudamqp-RabbitMQ- Loud Lion- 2 nodes	1.0	0.853	0.64	0.5	0.737	3499 $			
		NFR template 1	1.0	0.905	0.816	0.75	0.831	16070	0.884		
	Besoins 2		35 pts	35pts	10 pts	10 pts	10 pts				
	template 2	ServiceSQL : Amazon db.r4.16xlarge \textit{rovisioned IOPS} (multiAZ) :12571.32	1.0	0.957	0.993	1.0	0.926	12571 $			
		FileAttente1 : SQS Standard :4	1.0	0.988	0.155	1.0	0.737	4 $			
		NFR template 2	1.0	0.973	0.574	1.0	0.831	12575	0.931		
	Besoins 3		10 pts	10 pts	10 pts	35 pts	35pts				
	template 3	ServiceSQL : Amazon db.r4.16xlarge Provisioned IOPS (multiAZ) :12571.32	1.0	0.957	0.993	1.0	0.926	12571 $			
		FileAttente1 : SQS Standard :4	1.0	0.988	0.155	1.0	0.737	4 $			
		NFR template 3	1.0	0.973	0.574	1.0	0.831	12575	0.895		
Solution de Base	Besoins 1		35 pts	10 pts	35 pts	10 pts	10 pts			40766 $	0.902
	template 1	ServiceSQL : Amazon db.r4.16xlarge \textit{agnetic} (Mutli) :12344.27	1.0	0.957	0.989	1.0	0.926	12344 $			
		FileAttente :cloudamqp - RabbitMQ- Loud Lion - 2 nodes	1.0	0.853	0.64	0.5	0.737	3499 $			
		NFR template 1	1.0	0.905	0.814	0.75	0.831	15843	0.883		
	Besoins 2		35 pts	35pts	10 pts	10 pts	10 pts				
	template 2	ServiceSQL : Amazon db.r4.16xlarge\textit{agnetic} (Mutli), :12344.27	1.0	0.957	0.989	1.0	0.926	12344 $			
		FileAttente1 : SQS Standard :4	1.0	0.988	0.155	1.0	0.737	4 $			
		NFR template 2	1.0	0.973	0.572	1.0	0.831	12348	0.930		
	Besoins 3		10 pts	10 pts	10 pts	35 pts	35pts				
	template 3	ServiceSQL : Amazon db.r4.16xlarge Provisioned IOPS (multiAZ)" :12571.32	1.0	0.957	0.993		0.926	12571 $			
		FileAttente1 : SQS Standard :4	1.0	0.988	0.155	1.0	0.737	4 $			
		NFR template 3	1.0	0.973	0.574	1.0	0.831	12575	0.895		

TABLE 6.23 – Sélection d'une architecture de templates pour un coût illimité

l'architecture reste inférieur au prix imposé pour cette dernière. Nous avons construit, pour valider notre approche, des templates de référence qui représentent les meilleures valeurs de NFR pouvant être obtenues pour chaque contrainte de prix. Ainsi, nous avons justifié pour chacune des contraintes de prix que notre approche a permis d'obtenir les meilleures valeurs de NFR par rapport aux besoins requis pour chaque template.

6.6 Conclusion

Dans ce chapitre, nous avons d'abord présenté notre analyse du processus de composition de services. Ensuite, nous avons modélisé formellement notre problème de sélection de composition de services. Enfin, nous avons présenté notre méthodologie de sélection de composition de services qui est basée sur un algorithme génétique permettant de générer des solutions de composition dans un temps raisonnable, tout en adaptant la sélection des plans de services en fonction des besoins des architectes, notamment en termes de coûts. Notre méthodologie a pour but de guider les architectes dans la sélection des plans de services composant leurs architectures et d'adapter la sélection à leurs besoins tout en contrôlant le coût opérationnel. Cette méthodologie permet de sélectionner un ensemble de templates de services appropriés pour une architecture applicative. Elle se base principalement sur les résultats de la sélection des services cloud obtenus dans le chapitre précédent.

Nous avons validé notre approche de sélection de la composition des services en utilisant deux méthodes différentes : (1) La première a consisté à évaluer chaque template séparément. A cet égard, nous avons utilisé différentes exigences non fonctionnelles pour les templates et différentes exigences non fonctionnelles pour les services composant ces templates. (2) La seconde correspond à la validation par architecture de templates. Nous avons réutilisé chacun des templates de la première méthode pour construire des architectures composées de plusieurs instances de templates avec des exigences différentes. La contrainte est que le coût global de l'ensemble des templates composant l'architecture ne doit pas dépasser un coût prédéterminé.

Nous soulignons que l'approche proposée dans cette étude peut être réutilisée pour tout type de composition de services. Cependant, elle reste dépendante de la pertinence de la validation de la sélection de services. Elle dépend également de la disponibilité d'architectes

experts du domaine pour attribuer des poids significatifs aux besoins NFR de chaque template composant l'architecture

Dans le prochain chapitre, nous présentons brièvement nos contributions, nous soulignons les limites de nos travaux de thèse, et nous présentons nos perspectives et les travaux futurs.

Chapitre 7

Conclusion Générale

7.1 Rappel du cadre et des objectifs de la thèse

La sélection des services consiste à évaluer et à choisir les services les plus adaptés aux besoins spécifiques des architectes auprès des fournisseurs de services. Quant à la composition des services, elle consiste à évaluer et à sélectionner un ensemble de services qui répondent aux besoins applicatifs des architectes auprès des fournisseurs de services. Dans les deux cas, cette sélection doit garantir, d'une part, les besoins opérationnels de l'entreprise, tels que le besoin de capacité de stockage et la localisation des services~~, etc~~. et, d'autre part, la qualité du déploiement des services ou des applications, par exemple la minimisation des coûts de déploiement avec des performances élevées et une sécurité garantie.

La sélection des services [34] [149] [150] et la sélection de la composition des services [3] [2] [151] sont des domaines de recherche actifs mais nous pensons qu'ils n'ont jamais été déployés pour les types de services intergiciels que nous considérons dans cette thèse. Nous cherchons à proposer une approche efficace permettant de sélectionner des services répondant aux besoins d'architectes pour une application particulière en tenant compte de contraintes liées à leur fonctionnement et de contraintes de coût d'exploitation. Pour cela, nous devons répondre à trois questions.

1. Quels sont les éléments clés permettant de comparer des services pour permettre leur sélection ?
2. Comment sélectionner des services intergiciels pour une architecture de microservices ?

3. Comment sélectionner la composition des services qui répondent le mieux aux besoins applicatifs des architectes ?

Ainsi, notre travail de recherche vise à fournir une sélection guidée de services et de combinaisons de services cloud qui répondent aux exigences de qualité de déploiement de services (QoD) et aux contraintes exprimées par les architectes.

7.2 Principales contributions

Pour répondre aux différents défis mentionnés ci-dessus, nous avons proposé les trois contributions suivantes :

7.2.1 Contribution 1 : une méthodologie pour sélectionner les éléments clés de comparaison et leurs relations

Afin de surmonter la première problématique, nous proposons une méthodologie pour identifier les éléments clés de comparaison et leurs relations pour la sélection des services cloud. Notre méthodologie est basée sur des données réelles disponibles auprès des fournisseurs de services et sur des benchmarks. Elle se déroule comme suit : **(1)** Tout d'abord, nous menons une enquête auprès des architectes afin de comprendre leurs exigences en matière de sélection de services cloud ; **(2)** Ensuite, nous identifions les questions qui intéressent les architectes et les classons en fonction de leur influence sur les NFR ; **(3)** Puis, nous identifions les attributs des plans des fournisseurs de services qui répondent aux questions des architectes et leur influence sur les NFR à partir de la documentation technique ; **(4)** Par la suite, nous examinons les travaux sur l'évaluation des performances du cloud afin de recueillir les attributs qui influencent les performances ; **(5)** Enfin, nous menons une étude empirique pour nous assurer que la liste des attributs que nous avons identifiés était complète et pour ajouter ceux qui manquaient. Cette méthode nous permet de croire qu'elle peut servir de base à un processus décisionnel solide qui fait totalement défaut aujourd'hui.

Les résultats de cette étude sont utilisés ultérieurement pour évaluer les services cloud et la composition des services cloud par type d'application. Tout d'abord, l'importance relative des attributs et des NFR devrait être déterminée en fonction des types d'application et l'exactitude des résultats devrait être validée par les architectes. Ensuite, sur la base du résultat précédent, une méthodologie est proposée pour évaluer les services cloud et la

composition des services cloud.

7.2.2 Contribution 2 : une méthodologie pour la sélection de services middleware basés sur une architecture de micro-services

Nous proposons une méthodologie pour comparer les plans des fournisseurs de services de middleware afin de sélectionner les services appropriés pour une architecture d'application. Nous nous appuyons sur les résultats de l'étude présentée dans la première contribution et sur les exigences des architectes en matière d'architecture d'application. Notre méthodologie repose sur : **(1)** la résolution des difficultés liées à l'incapacité de mesurer l'impact des NFA sur les performances ; **(2)** la réalisation d'une enquête pour attribuer des poids significatifs aux NFA et aux NFR en fonction des besoins des architectes en matière d'applications ; **(3)** la normalisation des valeurs des NFA en fonction des types de données de ces derniers à l'aide de méthodes de normalisation adéquates ; **(4)** l'utilisation de la méthode d'aide à la décision *Ordered weighted averaging aggregation operator (OWA)* pour la classification des services cloud en fonction des besoins des architectes. Cette méthode de décision résout le problème de la compensation, éliminant ainsi des solutions très insatisfaisantes sur un NFA ou un NFR et **(5)** la proposition d'un algorithme de sélection de services qui utilise les valeurs NFA normalisées et la méthode de prise décision OWA.

Nous avons validé l'efficacité de notre framework de sélection des services cloud sur la base d'études de cas d'utilisation. Nous avons utilisé les configurations des plans des services et les avis d'experts dans le domaine pour démontrer que notre approche fournit les services adaptés aux besoins applicatifs des architectes.

7.2.3 Contribution 3 : une méthodologie pour la sélection de la composition des services en fonction du type d'application

Nous avons fourni une méthodologie pour guider les architectes dans la sélection des compositions de plans de services pour leurs architectures. Cette méthodologie repose sur : (1) les résultats de l'étude de sélection des services et sur l'analyse du processus de composition de services pour une architecture technique ; (2) un processus de composition de services, basé sur l'utilisation de l'algorithme génétique, qui permet de générer des solutions de compositions dans un délai raisonnable, tout en adaptant la sélection des plans de service en fonction des besoins des architectes, notamment en termes de coûts.

Nous avons validé la pertinence de notre approche de composition de services et l'avons appliqué à la sélection de services cloud composés de deux types de services : les services cloud de bases de données relationnelles et les services cloud file de messages. Nous avons utilisé deux méthodes différentes pour cette évaluation : (i) La première consiste à évaluer chaque Template séparément. À cet égard, nous avons utilisé différentes exigences non fonctionnelles pour les Templates et différentes exigences non fonctionnelles pour les services constituant ces Templates. Nous avons utilisé également deux contraintes. La première contrainte est que les services constituant les Templates doivent être basés dans une région spécifique, par exemple en Europe. La deuxième contrainte est que le coût global des services constituant le Template ne doit pas dépasser un montant prédéterminé ; (ii) La deuxième méthode de validation consiste à valider des architectures composées de plusieurs templates. Pour ce faire, nous avons réutilisé chacune des exigences des templates de la première méthode pour construire des architectures techniques. La contrainte à respecter est que le coût global de l'ensemble des templates constituant l'architecture ne doit pas dépasser un montant prédéterminé.

7.3 Limites de ces travaux de thèse

Nos contributions ne prétendent pas apporter une réponse parfaite et indiscutable aux différentes problématiques de thèse.

— **Limites liées à l'identification des éléments clés de comparaison et leurs relations**

La méthode que nous proposons peut être réutilisée pour tout type de service. Cependant, elle reste dépendante de la disponibilité des données (NFA) mises à disposition par les fournisseurs de services et des benchmarks disponibles dans la littérature. De plus, la pertinence des NFA et leur influence sur les NFR sont conditionnées par l'expertise des architectes. Par exemple, concernant la pertinence des VPCs pour assurer la sécurité des services de bases de données relationnelles, certains architectes pensent qu'il est indispensable de changer de fournisseur en l'absence de VPCs, d'autres pensent que cette pertinence est importante sans pour autant les obliger à changer de fournisseur et à migrer leurs services. Ainsi, la présence des NFA et leur influence sur les NFR dépend de la vision des architectes.

Nous avons besoin du plus grand nombre d'architectes possible pour prendre des décisions cohérentes.

— **Limites liées à la sélection des services cloud**

Nous identifions deux limitations principales liées à l'approche de sélection des services : la première concerne le fait que notre approche de sélection des services cloud repose sur les résultats des éléments de comparaison clés et de leurs relations obtenus précédemment. Ainsi, elle reste dépendante de la pertinence des NFA et NFR et de leur relation récoltée dans le chapitre 4. De plus, l'attribution de poids significatifs à l'influence des NFA sur les NFR dépend de la disponibilité des architectes experts du domaine et de leur niveau d'expertise. Par conséquent, il est nécessaire d'avoir autant d'architectes que possible pour vérifier la pertinence des poids attribués aux influences des NFA sur les NFR.

La deuxième concerne la validation de l'approche de sélection des services. Cette approche a été validée sur une base de données SQL et un service de file de messages pour différentes capacités et exigences de service. Nous l'avons ensuite vérifiée avec sept architectes du domaine. Nous pensons que cette approche nécessite une validation supplémentaire avec un plus grand nombre d'architectes. En outre, elle doit être validée sur d'autres types de services pour garantir la pertinence de ses résultats.

— **Limites liées à la sélection de la composition des services**

Nous identifions quelques limitations principales liées à l'approche de composition de services :

La première réside dans le fait que notre approche de composition de services cloud reste dépendante de la pertinence des résultats de sélection de services. En effet, la solution de composition de services s'appuie sur les résultats de sélection de services cloud obtenus précédemment.

La seconde concerne la validité architecturale avec seulement deux types de services : les services de base de données relationnelle et le service de file de messages. Cependant, nous devons considérer plus de types de services provenant d'un plus grand nombre de fournisseurs de services. Ainsi, nous pourrons non seulement voir comment les NFR varient en fonction du prix selon les fournisseurs de services, mais également confirmer nos conclusions sur la variation du prix par rapport aux NFR

des services.

La troisième concerne la validation menée uniquement par rapport à la configuration des services sans prendre en compte l'avis des architectes sur la composition des services. Or, notre solution nécessite plus de tests et d'avis d'architectes experts comme nous l'avons fait pour la solution de sélection des services.

7.4 Perspectives

Compte tenu des limites évoquées, nous avons dégagé un ensemble de perspectives. Ces perspectives nous les avons classées en trois catégories : les perspectives liées à l'identification des éléments clés de comparaison et leurs relations, les perspectives liées à la sélection des services et les perspectives liées à la composition des services.

— **Perspectives liées à l'identification des éléments clés de comparaison et leurs relations**

Afin d'améliorer la méthode proposée, les perspectives suivantes peuvent être envisagées :

1. demandez aux fournisseurs de services de communiquer des mesures uniformisées à des fins de comparaison. Par exemple, leur demander de fournir le débit et la latence de leurs plans de services selon les mêmes scénarios expérimentaux et sous les mêmes contraintes. Ainsi, il serait possible d'évaluer les performances des plans des services.
2. élargir les discussions avec les architectes pour obtenir plus d'informations sur les NFA et leurs influence sur les NFR.
3. réappliquer notre méthode pour différents types de services tels que les services d'apprentissage automatique.

— **Perspectives liées à la sélection des services**

1. Dans cette thèse, nous avons proposé une approche de sélection de services et nous l'avons validée sur deux types de services : les services cloud de bases de données relationnelles et les services cloud de file de messages. Les résultats de cette approche ont été validés auprès de 7 experts du domaine. Cette approche est intéressante, cependant, elle doit être validée par un plus grand nombre d'experts. De plus, pour garantir son efficacité, cette approche doit être validée pour

d'autres types de services tels que les services d'apprentissage automatique. Enfin, nous pouvons intégrer un système de monitoring afin de garantir les résultats obtenus pour la performance de la sélection de services.

2. Dans cette thèse, nous nous sommes concentrés sur la sélection des services en connaissant les spécifications techniques des services que les architectes veulent déployer pour leur application. Par exemple, nous savons que les architectes veulent déployer une base de données relationnelle. Ainsi, nous sommes intéressés par la comparaison des plans de service de base de données relationnelle offerts par différents fournisseurs de services. L'un des défis consiste à comparer les spécifications techniques des services. Par exemple, si les architectes souhaitent un service de stockage, quels types de services doivent être proposés ? Il faut envisager un service de base de données NoSQL ou des bases de données relationnelles traditionnelles.
3. La comparaison des coûts de déploiement des services cloud selon différentes stratégies de tarification devient un problème très difficile. L'un des défis consiste à présenter une approche qui aide les architectes cloud à trouver les services les plus rentables pour leurs besoins applicatifs à long terme dans le cadre de différentes stratégies de tarification.

— **Perspectives liées à la composition des services**

1. Dans cette thèse, nous avons utilisé deux approches pour valider notre méthode de composition de services : la première consiste à effectuer une validation basée sur un seul template et la seconde consiste à effectuer une validation sur une architecture composée de plusieurs templates. Pour la première approche, nous n'avons utilisé que deux types de services : les services de bases de données relationnelles et le service de file de messages. Il serait plus approprié, dans un travail futur, de considérer d'autres types de services et de les valider sur un plus grand nombre de fournisseurs de services. Quant à la deuxième approche, nous n'avons utilisé que trois templates par architecture pour la validation. Il serait plus approprié de valider notre approche en utilisant une quinzaine de micro-services avec un nombre plus important et différents types de services.
2. Il faudrait faire appel à des experts du domaine pour déterminer les différentes

exigences des services et des templates au sein des architectures techniques. En outre, il faudrait que ces experts procèdent à une validation plus complète que celle que nous avons effectuée, à savoir la simple validation par rapport à la configuration des services et des templates composant les architectures.

3. Notre travail de thèse s'est davantage concentré sur les questions liées au traitement des critères de qualité de déploiement des services et leur pertinence pour la sélection de la composition de services cloud que sur la question de la minimisation du temps d'exécution de la composition de services. Un travail futur serait de valider notre méthode de composition de services pour différentes architectures regroupant une quinzaine de micro-services, chacun ayant plusieurs services différents afin d'en déduire ses performances et de l'améliorer en fonction des résultats.

Bibliographie

[1] Ajith Singh and M Hemalatha. Cloud computing for academic environment. 2012.

[2] Siva Kumar Gavvala, Chandrashekar Jatoth, GR Gangadharan, and Rajkumar Buyya. Qos-aware cloud service composition using eagle strategy. Future Generation Computer Systems, 90 :273–290, 2019.

[3] Amin Jula, Elankovan Sundararajan, and Zalinda Othman. Cloud computing service composition : A systematic literature review. Expert systems with applications, 41(8) :3809–3824, 2014.

[4] Fateh Seghir and Abdellah Khababa. A hybrid approach using genetic and fruit fly optimization algorithms for qos-aware cloud service composition. Journal of Intelligent Manufacturing, 29(8) :1773–1792, 2018.

[5] Tomas Cerny, Michael J Donahoo, and Michal Trnka. Contextual understanding of microservice architecture : current and future directions. ACM SIGAPP Applied Computing Review, 17(4) :29–45, 2018.

[6] Laura Martínez Garcia, Silvana Aciar, Raynel Mendoza, and Juan José Puello. Smart tourism platform based on microservice architecture and recommender services. In International Conference on Mobile Web and Intelligent Information Systems, pages 167–180. Springer, 2018.

[7] Zibin Zheng, Xinmiao Wu, Yilei Zhang, Michael R Lyu, and Jianmin Wang. Qos ranking prediction for cloud services. IEEE transactions on parallel and distributed systems, 24(6) :1213–1222, 2012.

[8] Raed Karim, Chen Ding, and Ali Miri. An end-to-end qos mapping approach for cloud service selection. In Services (SERVICES), 2013 IEEE Ninth World Congress on, pages 341–348. IEEE, 2013.

[9] Thiruselvan Subramanian and Nickolas Savarimuthu. Cloud service evaluation and

selection using fuzzy hybrid mcdm approach in marketplace. International Journal of Fuzzy System Applications (IJFSA), 5(2) :118–153, 2016.

[10] Philipp Leitner and Jürgen Cito. Patterns in the chaos—a study of performance variation and predictability in public iaas clouds. ACM Transactions on Internet Technology (TOIT), 16(3) :15, 2016.

[11] Cheol-Rim Choi and Hwa-Young Jeong. Quality evaluation and best service choice for cloud computing based on user preference and weights of attributes using the analytic network process. Electronic Commerce Research, 14(3) :245–270, 2014.

[12] J Octavio Gutierrez-Garcia and Kwang-Mong Sim. Agent-based service composition in cloud computing. In Grid and distributed computing, control and automation, pages 1–10. Springer, 2010.

[13] Daniel Worm, Miroslav Živković, Hans van den Berg, and Rob van der Mei. Revenue maximization with quality assurance for composite web services. In 2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pages 1–9. IEEE, 2012.

[14] Qi Yu and Athman Bouguettaya. Efficient service skyline computation for composite service selection. IEEE Transactions on Knowledge and Data Engineering, 25(4) :776–789, 2011.

[15] Erik Wittern, Jörn Kuhlenkamp, and Michael Menzel. Cloud service selection based on variability modeling. In International Conference on Service-Oriented Computing, pages 127–141. Springer, 2012.

[16] Mohan Baruwal Chhetri, Sergei Chichin, Quoc Bao Vo, and Ryszard Kowalczyk. Smart cloudbench—a framework for evaluating cloud infrastructure performance. Information Systems Frontiers, 18(3) :413–428, 2016.

[17] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp : comparing public cloud providers. In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, pages 1–14. ACM, 2010.

[18] Enrico Bocchi, Marco Mellia, and Sofiane Sarni. Cloud storage service benchmarking : Methodologies and experimentations. In 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), pages 395–400. IEEE, 2014.

[19] Xianrong Zheng, Patrick Martin, Kathryn Brohman, and Li Da Xu. Cloudqual : a quality model for cloud services. IEEE transactions on industrial informatics, 10(2) :1527–1536, 2014.

[20] Shyam S Wagle, Mateusz Guzek, Pascal Bouvry, and Raymond Bisdorff. An evaluation model for selecting cloud services from commercially available cloud providers. In Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on, pages 107–114. IEEE, 2015.

[21] Wayne A Jansen, Tim Grance, et al. Guidelines on security and privacy in public cloud computing. 2011.

[22] Christina N Hoefer and Georgios Karagiannis. Taxonomy of cloud computing services. In 2010 IEEE Globecom Workshops, pages 1345–1350. IEEE, 2010.

[23] Michael Hogan, Fang Liu, Annie Sokol, and Jin Tong. Nist cloud computing standards roadmap. NIST Special Publication, 35 :6–11, 2011.

[24] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. A framework for ranking of cloud computing services. Future Generation Computer Systems, 29(4) :1012–1023, 2013.

[25] David Villegas and Seyed Masoud Sadjadi. Mapping non-functional requirements to cloud applications. In SEKE, pages 527–532, 2011.

[26] Shoaib Hassan, Farooque Azam, et al. Analysis of cloud computing performance, scalability, availability, & security. In 2014 International Conference on Information Science & Applications (ICISA), pages 1–5. IEEE, 2014.

[27] Juan Cáceres, Luis M Vaquero, Luis Rodero-Merino, Álvaro Polo, and Juan J Hierro. Service scalability over the cloud. In Handbook of Cloud Computing, pages 357–377. Springer, 2010.

[28] Joseph Idziorek. Discrete event simulation model for analysis of horizontal scaling in the cloud computing model. In Proceedings of the 2010 Winter Simulation Conference, pages 3004–3014. IEEE, 2010.

[29] Shailesh Paliwal. Performance challenges in cloud computing, 2014.

[30] Gartner. Gartner forecasts worldwide public cloud revenue to grow 17% in 2020. https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2020.

[31] Vadim Vladimirskiy. 10 popular software as a service (saas) examples. https://getnerdio.com/academy/10-popular-software-service-examples/.

[32] Mario Villamizar, Oscar Garcés, Harold Castro, Mauricio Verano, Lorena Salamanca, Rubby Casallas, and Santiago Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In 2015 10th Computing Colombian Conference (10CCC), pages 583–590. IEEE, 2015.

[33] Channu Kambalyal. 3-tier architecture. Retrieved On, 2 :34, 2010.

[34] Le Sun, Hai Dong, Farookh Khadeer Hussain, Omar Khadeer Hussain, and Elizabeth Chang. Cloud service selection : State-of-the-art and future research directions. Journal of Network and Computer Applications, 45 :134–150, 2014.

[35] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software : Practice and experience, 41(1) :23–50, 2011.

[36] Henri Casanova. Simgrid : A toolkit for the simulation of application scheduling. In Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 430–437. IEEE, 2001.

[37] Gülçin Büyüközkan and Gizem Çifçi. A novel hybrid mcdm approach based on fuzzy dematel, fuzzy anp and fuzzy topsis to evaluate green suppliers. Expert Systems with Applications, 39(3) :3000–3011, 2012.

[38] Umut Asan, Ayberk Soyer, and Seyda Serdarasan. A fuzzy analytic network process approach. In Computational intelligence systems in industrial engineering, pages 155–179. Springer, 2012.

[39] Mehmet Sevkli. An application of the fuzzy electre method for supplier selection. International Journal of Production Research, 48(12) :3393–3405, 2010.

[40] Jane Siegel and Jeff Perdue. Cloud services measures for global use : the service measurement index (smi). In 2012 Annual SRII global conference, pages 411–415. IEEE, 2012.

[41] Sun Le, Hai Dong, Farookh Khadeer Hussain, Omar Khadeer Hussain, Jiangang Ma, and Yanchun Zhang. Multicriteria decision making with fuzziness and criteria

interdependence in cloud service selection. In 2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pages 1929–1936. IEEE, 2014.

[42] Quanwang Wu, Qingsheng Zhu, and Mingqiang Zhou. A correlation-driven optimal service selection approach for virtual enterprise establishment. Journal of Intelligent Manufacturing, 25(6) :1441–1453, 2014.

[43] Yuli Yang, Nanyue Yu, and Yongle Chen. Trusted cloud service selection algorithm based on lightweight intuitionistic fuzzy numbers. IEEE Access, 8 :97748–97756, 2020.

[44] Abdullah Al-Faifi, Biao Song, Mohammad Mehedi Hassan, Atif Alamri, and Abdu Gumaei. A hybrid multi criteria decision method for cloud service selection from smart data. Future Generation Computer Systems, 93 :43–57, 2019.

[45] Alexander Lenk, Michael Menzel, Johannes Lipsky, Stefan Tai, and Philipp Offermann. What are you paying for ? performance benchmarking for infrastructure-as-a-service offerings. In 2011 IEEE 4th International Conference on Cloud Computing, pages 484–491. IEEE, 2011.

[46] Chunjie Luo, Jianfeng Zhan, Zhen Jia, Lei Wang, Gang Lu, Lixin Zhang, Cheng-Zhong Xu, and Ninghui Sun. Cloudrank-d : benchmarking and ranking cloud computing systems for data processing applications. Frontiers of Computer Science, 6(4) :347–362, 2012.

[47] Alexandru Iosup, Radu Prodan, and Dick Epema. Iaas cloud benchmarking : approaches, challenges, and experience. In Cloud Computing for Data-Intensive Applications, pages 83–104. Springer, 2014.

[48] Alexandru Iosup, Simon Ostermann, M Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick Epema. Performance analysis of cloud computing services for many-tasks scientific computing. IEEE Transactions on Parallel and Distributed systems, 22(6) :931–945, 2011.

[49] Jiang Dejun, Guillaume Pierre, and Chi-Hung Chi. Ec2 performance analysis for resource provisioning of service-oriented applications. In Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops, pages 197–207. Springer, 2009.

[50] Amir Hossein Borhani, Philipp Leitner, Bu-Sung Lee, Xiaorong Li, and Terence Hung. Wpress : An application-driven performance benchmark for cloud-based

virtual machines. In 2014 IEEE 18th International Enterprise Distributed Object Computing Conference, pages 101–109. IEEE, 2014.

[51] Jörg Schad, Jens Dittrich, and Jorge-Arnulfo Quiané-Ruiz. Runtime measurements in the cloud : observing, analyzing, and reducing variance. Proceedings of the VLDB Endowment, 3(1-2) :460–471, 2010.

[52] Benjamin Farley, Ari Juels, Venkatanathan Varadarajan, Thomas Ristenpart, Kevin D Bowers, and Michael M Swift. More for your money : exploiting performance heterogeneity in public clouds. In Proceedings of the Third ACM Symposium on Cloud Computing, pages 1–14, 2012.

[53] Zhonghong Ou, Hao Zhuang, Andrey Lukyanenko, Jukka K Nurminen, Pan Hui, Vladimir Mazalov, and Antti Ylä-Jääski. Is the same instance type created equal ? exploiting heterogeneity of public clouds. IEEE transactions on cloud computing, 1(2) :201–214, 2013.

[54] Davide Cerotti, Marco Gribaudo, Pietro Piazzolla, and Giuseppe Serazzi. Flexible cpu provisioning in clouds : A new source of performance unpredictability. In 2012 Ninth International Conference on Quantitative Evaluation of Systems, pages 230–237. IEEE, 2012.

[55] Sean Kenneth Barker and Prashant Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In Proceedings of the first annual ACM SIGMM conference on Multimedia systems, pages 35–46, 2010.

[56] Guohui Wang and TS Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center. In 2010 Proceedings IEEE INFOCOM, pages 1–9. IEEE, 2010.

[57] Lars Kotthoff. Reliability of computational experiments on virtualised hardware. Journal of Experimental & Theoretical Artificial Intelligence, 26(1) :33–49, 2014.

[58] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. Decision support systems, 43(2) :618–644, 2007.

[59] Lie Qu et al. Credible service selection in cloud environments. 2016.

[60] Ziqiang Xu, Patrick Martin, Wendy Powley, and Farhana Zulkernine. Reputation-enhanced qos-based web services discovery. In IEEE International Conference on Web Services (ICWS 2007), pages 249–256. IEEE, 2007.

[61] Servicetrust : Supporting reputation-oriented service selection.

[62] Zeinab Noorian, Michael Fleming, and Stephen Marsh. Preference-oriented qos-based service discovery with dynamic trust and reputation management. In Proceedings of the 27th Annual ACM Symposium on Applied Computing, pages 2014–2021, 2012.

[63] Zia ur Rehman, Omar K Hussain, Sazia Parvin, and Farookh K Hussain. A framework for user feedback based cloud service monitoring. In 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems, pages 257–262. IEEE, 2012.

[64] Talal H Noor and Quan Z Sheng. Trust as a service : A framework for trust management in cloud environments. In International conference on web information systems engineering, pages 314–321. Springer, 2011.

[65] Talal H Noor and Quan Z Sheng. Credibility-based trust management for services in cloud environments. In International Conference on Service-Oriented Computing, pages 328–343. Springer, 2011.

[66] Shuai Ding, Shanlin Yang, Youtao Zhang, Changyong Liang, and Chengyi Xia. Combining qos prediction and customer satisfaction estimation to solve cloud service trustworthiness evaluation problems. Knowledge-Based Systems, 56 :216–225, 2014.

[67] Lie Qu, Yan Wang, and Mehmet A Orgun. Cloud service selection based on the aggregation of user feedback and quantitative performance assessment. In 2013 IEEE International Conference on Services Computing, pages 152–159. IEEE, 2013.

[68] Lie Qu, Yan Wang, Mehmet A Orgun, Ling Liu, and Athman Bouguettaya. Cloud service selection based on contextual subjective assessment and objective assessment. In Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, pages 1483–1484, 2014.

[69] Lie Qu, Yan Wang, Mehmet A Orgun, Ling Liu, and Athman Bouguettaya. Context-aware cloud service selection based on comparison and aggregation of user subjective assessment and objective performance assessment. In 2014 IEEE International Conference on Web Services, pages 81–88. IEEE, 2014.

[70] Paolo Massa and Paolo Avesani. Trust metrics on controversial users : Balancing between tyranny of the majority. International Journal on Semantic Web and Information Systems (IJSWIS), 3(1) :39–64, 2007.

[71] Zhu Yong, Li Wei, Luo Junzhou, and Zheng Xiao. A novel two-phase approach for qos-aware service composition based on history records. In 2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pages 1–8. IEEE, 2012.

[72] Wenying Zeng, Yuelong Zhao, and Junwei Zeng. Cloud service and service selection algorithm research. In Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation, pages 1045–1048. 2009.

[73] Kevin Kofler, Irfan ul Haq, and Erich Schikuta. A parallel branch and bound algorithm for workflow qos optimization. In 2009 International Conference on Parallel Processing, pages 478–485. IEEE, 2009.

[74] Sheng Liu, Gang Xiong, Hongxia Zhao, Xisong Dong, and Jianshi Yao. Service composition execution optimization based on state transition matrix for cloud computing. In Proceedings of the 10th World Congress on Intelligent Control and Automation, pages 4126–4131. IEEE, 2012.

[75] Xinchao Zhao, Zichao Wen, and Xingmei Li. Qos-aware web service selection with negative selection algorithm. Knowledge and information systems, 40(2) :349–373, 2014.

[76] Simone A Ludwig. Clonal selection based genetic algorithm for workflow service selection. In 2012 IEEE Congress on Evolutionary Computation, pages 1–7. IEEE, 2012.

[77] Alexander Jungmann and Bernd Kleinjohann. Towards the application of reinforcement learning techniques for quality-based service selection in automated service composition. In 2012 IEEE Ninth International Conference on Services Computing, pages 701–702. IEEE, 2012.

[78] Shangguang Wang, Qibo Sun, Hua Zou, and Fangchun Yang. Particle swarm optimization with skyline operator for fast cloud-based web service composition. Mobile Networks and Applications, 18(1) :116–121, 2013.

[79] Zhen Ye, Xiaofang Zhou, and Athman Bouguettaya. Genetic algorithm based qos-aware service compositions in cloud computing. In International Conference on Database Systems for Advanced Applications, pages 321–334. Springer, 2011.

[80] Huihui Bao and Wanchun Dou. A qos-aware service selection method for cloud service

composition. In 2012 IEEE 26th international parallel and distributed processing symposium workshops & Phd Forum, pages 2254–2261. IEEE, 2012.

[81] Smitha Sundareswaran, Anna Squicciarini, and Dan Lin. A brokerage-based approach for cloud service selection. In 2012 IEEE Fifth International Conference on Cloud Computing, pages 558–565. IEEE, 2012.

[82] Qingtao Wu, Mingchuan Zhang, Ruijuan Zheng, Ying Lou, and Wangyang Wei. A qos-satisfied prediction model for cloud-service composition based on a hidden markov model. Mathematical Problems in Engineering, 2013, 2013.

[83] Tran Vu Pham, Hani Jamjoom, Kirk Jordan, and Zon-Yin Shae. A service composition framework for market-oriented high performance computing cloud. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pages 284–287, 2010.

[84] Miranda Zhang, Rajiv Ranjan, Surya Nepal, Michael Menzel, and Armin Haller. A declarative recommender system for cloud infrastructure services selection. In International Conference on Grid Economics and Business Models, pages 102–113. Springer, 2012.

[85] Asrin Vakili and Nima Jafari Navimipour. Comprehensive and systematic review of the service composition mechanisms in the cloud environments. Journal of Network and Computer Applications, 81 :24–36, 2017.

[86] Han Zhang, Rui Feng Guo, and Cong Geng. Hierarchical optimization model of cloud manufacturing services combination. In Applied Mechanics and Materials, volume 464, pages 345–351. Trans Tech Publ, 2014.

[87] Biqing Huang, Chenghai Li, and Fei Tao. A chaos control optimal algorithm for qos-based service composition selection in cloud manufacturing system. Enterprise Information Systems, 8(4) :445–463, 2014.

[88] Guisheng Fan, Huiqun Yu, Liqiong Chen, and Dongmei Liu. Petri net based techniques for constructing reliable service composition. Journal of Systems and Software, 86(4) :1089–1106, 2013.

[89] Aarti Singh, Dimple Juneja, and Manisha Malhotra. A novel agent based autonomous and service composition framework for cost optimization of resource provisioning

in cloud computing. Journal of King Saud University-Computer and Information Sciences, 29(1) :19–28, 2017.

[90] Djamel Benmerzoug, Mohamed Gharzouli, and Mounira Zerari. Agent interaction protocols in support of cloud services composition. In Industrial Applications of Holonic and Multi-Agent Systems, pages 293–304. Springer, 2013.

[91] Dragan Ivanović and Manuel Carro. Transforming service compositions into cloud-friendly actor networks. In International Conference on Service-Oriented Computing, pages 291–305. Springer, 2014.

[92] Xin Zhao, Liwei Shen, Xin Peng, and Wenyun Zhao. Toward sla-constrained service composition : An approach based on a fuzzy linguistic preference model and an evolutionary algorithm. Information Sciences, 316 :370–396, 2015.

[93] Adrian Klein, Fuyuki Ishikawa, and Shinichi Honiden. Sanga : A self-adaptive network-aware approach to service composition. IEEE Transactions on Services Computing, 7(3) :452–464, 2013.

[94] Yongkui Liu, Lihui Wang, Yuquan Wang, Xi Vincent Wang, and Lin Zhang. Multi-agent-based scheduling in cloud manufacturing with dynamic task arrivals. Procedia CIRP, 72.

[95] Ying Huo, Yi Zhuang, Jingjing Gu, Siru Ni, and Yu Xue. Discrete gbest-guided artificial bee colony algorithm for cloud service composition. Applied Intelligence, 42(4) :661–678, 2015.

[96] Mohammad Bagher Karimi, Ayaz Isazadeh, and Amir Masoud Rahmani. Qos-aware service composition in cloud computing using data mining techniques and genetic algorithm. The Journal of Supercomputing, 73(4) :1387–1415, 2017.

[97] Hisham A Kholidy, Hala Hassan, Amany M Sarhan, Abdelkarim Erradi, and Sherif Abdelwahed. Qos optimization for cloud service composition based on economic model. In International Internet of Things Summit, pages 355–366. Springer, 2014.

[98] Heba Kurdi, Abeer Al-Anazi, Carlene Campbell, and Auhood Al Faries. A combinatorial optimization algorithm for multiple cloud service composition. Computers & Electrical Engineering, 42 :107–113, 2015.

[99] Haithem Mezni and Mokhtar Sellami. Multi-cloud service composition using formal concept analysis. Journal of Systems and Software, 134 :138–152, 2017.

[100] Fatma Lahmar and Haithem Mezni. Multicloud service composition : a survey of current approaches and issues. Journal of Software : Evolution and Process, 30(10) :e1947, 2018.

[101] Narges Hesami Rostami, Esmaeil Kheirkhah, and Mehrdad Jalali. An optimized semantic web service composition method based on clustering and ant colony algorithm. arXiv preprint arXiv :1402.2271, 2014.

[102] Qiang Yu, Ling Chen, and Bin Li. Ant colony optimization applied to web service compositions in cloud computing. Computers & Electrical Engineering, 41 :18–27, 2015.

[103] Simone A Ludwig. Applying particle swarm optimization to quality-of-service-driven web service composition. In 2012 IEEE 26th international conference on advanced information networking and applications, pages 613–620. IEEE, 2012.

[104] Dandan Wang, Yang Yang, and Zhenqiang Mi. A genetic-based approach to web service composition in geo-distributed cloud environment. Computers & Electrical Engineering, 43 :129–141, 2015.

[105] Guobing Zou, Yixin Chen, Y Yang, Ruoyun Huang, and You Xu. Ai planning and combinatorial optimization for web service composition in cloud computing. In Annual International Conference on Cloud Computing and Virtualization (CCV 2010), page 28, 2010.

[106] Fan Zhang, Kai Hwang, Samee U Khan, and Qutaibah M Malluhi. Skyline discovery and composition of multi-cloud mashup services. IEEE Transactions on Services Computing, 9(1) :72–83, 2015.

[107] Wanchun Dou, Xuyun Zhang, Jianxun Liu, and Jinjun Chen. Hiresome-ii : Towards privacy-aware cross-cloud service composition for big data applications. IEEE Transactions on Parallel and Distributed Systems, 26(2) :455–466, 2013.

[108] J Octavio Gutierrez-Garcia and Kwang Mong Sim. Agent-based cloud service composition. Applied intelligence, 38(3) :436–464, 2013.

[109] Kyriakos Kritikos and Dimitris Plexousakis. Multi-cloud application design through cloud service composition. In 2015 IEEE 8th international conference on cloud computing, pages 686–693. IEEE, 2015.

[110] Miao Zhang, Li Liu, and Songtao Liu. Genetic algorithm based qos-aware service composition in multi-cloud. In 2015 IEEE Conference on Collaboration and Internet Computing (CIC), pages 113–118. IEEE, 2015.

[111] Rong Yang, Bing Li, Jian Wang, Lulu He, and Xiaohui Cui. Scky : A method for reusing service process fragments. In 2014 IEEE International Conference on Web Services, pages 209–216. IEEE, 2014.

[112] Amazon rds multi-az deployments. https://aws.amazon.com/rds/details/multi-az/.

[113] 2ndwatch. Benchmarking amazon aurora. https://www.2ndwatch.com/blog/benchmarking-amazon-aurora/, 6 September 2016.

[114] David Ameller, Claudia Ayala, Jordi Cabot, and Xavier Franch. Non-functional requirements in architectural decision making. IEEE software, 30(2) :61–67, 2012.

[115] Rafael Capilla, Anton Jansen, Antony Tang, Paris Avgeriou, and Muhammad Ali Babar. 10 years of software architecture knowledge management : Practice and future. Journal of Systems and Software, 116 :191–205, 2016.

[116] Comparison of mysql across aws, azure and gcp. https://medium.com/@lakshmanLD/comparison-of-mysql-across-aws-azure-and-gcp-19af2d208d9a, 3 February 2018.

[117] Cloud sql for mysql documentation. https://cloud.google.com/sql/docs/mysql/, 21 June 2018.

[118] Using oracle mysql cloud service. https://docs.oracle.com/cloud/latest/mysql-cloud/UOMCS/UOMCS.pdf/, 2018.

[119] Philip A Bernstein, Istvan Cseri, Nishant Dani, Nigel Ellis, Ajay Kalhan, Gopal Kakivaya, David B Lomet, Ramesh Manne, Lev Novik, and Tomas Talius. Adapting microsoft sql server for cloud computing. In 2011 IEEE 27th International Conference on Data Engineering, pages 1255–1263. IEEE, 2011.

[120] Willis Lang, Frank Bertsch, David J DeWitt, and Nigel Ellis. Microsoft azure sql database telemetry. In Proceedings of the Sixth ACM Symposium on Cloud Computing, pages 189–194. ACM, 2015.

[121] ascent Thought leadership from Atos. Cloud messaging : Extending

cloud orchestration. https://atos.net/wp-content/uploads/2017/10/01042014-AscentWhitePaper-CloudMessaging.pdf.

[122] Storage queues and service bus queues - compared and contrasted. https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted, 4 september, 2019.

[123] Comparing apache kafka, amazon kinesis, microsoft event hubs and google pub/sub. https://blog.scottlogic.com/2018/04/17/comparing-big-data-messaging.html, 17 April 2018.

[124] LOVISA JOHANSSON. Rabbitmq best practices. https://www.cloudamqp.com/blog/2017-12-29-part1-rabbitmq-best-practice.html.

[125] Adam Warski. Benchmarking message queue latency. https://bravenewgeek.com/benchmarking-message-queue-latency/, 13 FEBRUARY 2016.

[126] EC Amazon. Amazon web services. Available in : http ://aws. amazon. com/es/ec2/(November 2012), 2015.

[127] Microsoft Azure. Azure cache for redis documentation. https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/?fbclid=IwAR16SezoBPQGa-b3twyF-uyWjx3Xu1_ft5jt1nSZ7UL049zg1QVTqszoWD8.

[128] Google. Documentation cloud memorystore pour redis. https://cloud.google.com/memorystore/docs/redis/.

[129] Microsoft Azure. Azure premium storage : design for high performance. https://docs.microsoft.com/en-us/azure/virtual-machines/windows/premium-storage-performance.

[130] Amazon AWS. Boosting application performance and reducing costs with amazon elasticache for redis. https://aws.amazon.com/fr/blogs/database/boosting-application-performance-and-reducing-costs-with-amazon-elasticache-for-redis/.

[131] Azure Microsoft. Azure cache for redis faq. https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-faq.

[132] Mingdong Tang, Xiaoling Dai, Jianxun Liu, and Jinjun Chen. Towards a trust evaluation middleware for cloud service selection. Future Generation Computer Systems, 74 :302–312, 2017.

[133] Athman Bouguettaya, Munindar Singh, Michael Huhns, Quan Z. Sheng, Hai Dong, Qi Yu, Azadeh Ghari Neiat, Sajib Mistry, Boualem Benatallah, Brahim Medjahed, Mourad Ouzzani, Fabio Casati, Xumin Liu, Hongbing Wang, Dimitrios Georgakopoulos, Liang Chen, Surya Nepal, Zaki Malik, Abdelkarim Erradi, Yan Wang, Brian Blake, Schahram Dustdar, Frank Leymann, and Michael Papazoglou. A service computing manifesto : The next 10 years. Commun. ACM, 60(4) :64–72, March 2017.

[134] Vicenç Torra. The weighted owa operator. International Journal of Intelligent Systems, 12(2) :153–166, 1997.

[135] Types d'instances amazon rds. https://aws.amazon.com/fr/rds/instance-types/.

[136] Overview of azure sql managed instance resource limits. https://docs.microsoft.com/en-us/azure/azure-sql/managed-instance/resource-limits.

[137] vpsbenchmarks. Microsoft azure vs amazon ec2. https://www.vpsbenchmarks.com/compare/azure_vs_ec2.

[138] B Delibašić, J Hernández, J Papathanasiou, F Dargam, I Linden, S Liu, R Ribeiro, and P Zaraté. Importance of data normalization in decision making : case study with topsis method.

[139] Priti Sudhir Patki and Vishakha V Kelkar. Classification using different normalization techniques in support vector machine. In International Conference on Communication Technology, pages 17–19, 2013.

[140] Ronald R Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. IEEE Transactions on systems, Man, and Cybernetics, 18(1) :183–190, 1988.

[141] Adam Warski. Evaluating persistent, replicated message queues. https://softwaremill.com/mqperf/, 18 July 2017.

[142] Fatma Lahmar and Haithem Mezni. Security-aware multi-cloud service composition by exploiting rough sets and fuzzy fca. Soft Computing, 25(7) :5173–5197, 2021.

[143] Huagang Liang, Xiaoqian Wen, Yongkui Liu, Haifeng Zhang, Lin Zhang, and Lihui Wang. Logistics-involved qos-aware service composition in cloud manufacturing with deep reinforcement learning. Robotics and Computer-Integrated Manufacturing, 67 :101991, 2021.

[144] Kouros Zanbouri and Nima Jafari Navimipour. A cloud service composition method using a trust-based clustering algorithm and honeybee mating optimization algorithm. International Journal of Communication Systems, 33(5) :e4259, 2020.

[145] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. IEEE Transactions on software engineering, 33(6) :369–384, 2007.

[146] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization : Overview and conceptual comparison. ACM computing surveys (CSUR), 35(3) :268–308, 2003.

[147] Darrell Whitley. A genetic algorithm tutorial. Statistics and computing, 4(2) :65–85, 1994.

[148] Binh Minh Nguyen, Trung Tran, Thieu Nguyen, and Giang Nguyen. Hybridization of galactic swarm and evolution whale optimization for global search problem. IEEE Access, 8 :74991–75010, 2020.

[149] Chandrashekar Jatoth, GR Gangadharan, Ugo Fiore, and Rajkumar Buyya. Selcloud : a hybrid multi-criteria decision-making model for selection of cloud services. Soft Computing, 23(13) :4701–4715, 2019.

[150] Somu Nivethitha, MR Gauthama Raman, Obulaporam Gireesha, Krithivasan Kannan, and VS Shankar Sriram. An improved rough set approach for optimal trust measure parameter selection in cloud environments. Soft Computing, 23(22) :11979–11999, 2019.

[151] Yongxiang Li, Xifan Yao, and Min Liu. Cloud manufacturing service composition optimization with improved genetic algorithm. Mathematical Problems in Engineering, 2019, 2019.

Annexe A

Questionnaire

A.1 Objectifs de la recherche

Nous cherchons à comprendre comment les architectes décident des types de composants qu'ils utilisent pour développer une nouvelle application et quels sont les critères qui guident leur choix. Nous nous intéressons particulièrement aux composants des intergiciels (base de données, messagerie, mise en cache, indexation et surveillance) qui peuvent être déployés comme serveurs ou consommés comme services cloud. Pour des raisons de confidentialité, les réponses à cette enquête ne peuvent être publiées.

— **Questions sur l'expérience des architectes**

1. Quelle est votre position actuelle ?
2. Depuis combien de temps occupez-vous ce poste ?
3. Décrivez votre expérience pour ce poste en tant qu'architecte ?
4. Dans quels types de projets avez-vous été impliqué ?
5. Quel était le personnel annuel moyen (à temps plein et à temps partiel) affecté au projet ?
6. Quelle était la date (approximative) de début du projet ?
7. Quand le projet a-t-il été entièrement livré pour la première fois ?
8. Quelle était la durée de la prise de décision ?

— **Questions concernant l'application globale**

1. Description de l'application
   (a) Décrivez votre application. Que fait-elle, qui va l'utiliser ?

(b) Décrivez son architecture technique (ses composants, les services qui la constituent)

(c) Quel est le modèle commercial de l'application ? (capacité, licence, utilisation...)

(d) Comment est-elle commercialisée ?

2. Besoins de l'application

(a) Quelles sont vos mesures de performance pour cette application ?

(b) Quels sont les besoins par rapport à ces mesures si vous les connaissez ? Évoluent-ils et si oui, selon quelle courbe ? augmentation ? linéaire ? évolution périodique, saisonnière ?

(c) Quels sont les besoins en matière de disponibilité ?

(d) Quels sont les besoins en matière de conservation des données ? (volume, fréquence, durée...)

(e) Quels sont vos besoins en matière de surveillance ? (fréquences, détails)

(f) Quels sont vos besoins en matière de journalisation ? (durée, centralisation des logs...)

3. Contraintes d'application

(a) Quelles sont les contraintes légales, réglementaires et de conformité ?

(b) Quelles sont vos contraintes en termes de sécurité ? accès, données... ?

(c) Quelles sont vos contraintes en termes de portabilité des données ?

(d) Avez-vous des contraintes spécifiques d'utilisation des services/outils dans votre architecture ? (problème d'interopérabilité)

(e) Avez-vous des contraintes de partenariat qui vous imposent des choix technologiques ?

(f) Avez-vous besoin de connecter votre service à des API spécifiques ? Quelles sont ces API ?

(g) Avez-vous des contraintes liées au développement ? [langage cadre...]

(h) Avez-vous une contrainte budgétaire pour l'exploitation ?

(i) Avez-vous un plan de reprise après un sinistre ? Quels sont vos besoins concernant les caractéristiques de cette propriété ?

— **Questions sur chacun des services/composants utilisés**

1. Le service doit-il se conformer à une norme ? si oui, laquelle ?
2. Avez-vous des exigences concernant la configuration de ce service ?
3. Avez-vous des besoins particuliers en matière de performance ou de capacité pour ce service ? Ces besoins varieront-ils dans le temps et, si oui, selon quelle période ou fréquence ?
4. Avez-vous des exigences particulières en matière de sécurité pour ce service ? Ce service est-il essentiel pour votre application ? Expliquez pourquoi il est critique ou pas.
5. Avez-vous des exigences particulières concernant l'utilisation du service pendant le développement ou l'exploitation ?
6. Quel type de support attendez-vous au moins ?
7. Quels sont vos besoins en matière de documentation (migration, référence API...) ? Pour quel service ?
8. Avez-vous rencontré des problèmes lors de l'utilisation des services ? Si oui, quels sont ces services ? Quels sont ces problèmes pour chaque service ? Comment les avez-vous résolus ?