Formalisation et Évaluation de Stratégies d'Élasticité Multi-couches dans le Cloud

Khaled Khebbeb

▶ **To cite this version:**

Khaled Khebbeb. Formalisation et Évaluation de Stratégies d'Élasticité Multi-couches dans le Cloud. Informatique [cs]. LIUPPA - Laboratoire Informatique de l'Université de Pau et des Pays de l'Adour; LIRE - Laboratoire d'Informatique Répartie de l'Université Constantine 2, 2019. Français. NNT : 2019PAUU3010 . tel-02271523

HAL Id: tel-02271523

https://theses.hal.science/tel-02271523v1

Submitted on 2 Mar 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR — LIUPPA — UNIVERSITÉ Abdelhamid MEHRI CONSTANTINE 2 — LIRE

Université de Pau et des Pays de l'Adour - École doctorale des Sciences Exactes et leurs Applications (ED SEA 211)
Université Constantine 2 Abdelhamid Mehri - Faculté des Nouvelles Technologies de l'Information et de la Communication - Département des Technologies Logicielles et Systèmes d'Information

Formalisation et évaluation de stratégies d'élasticité multi-couches dans le Cloud

THÉSE

pour l'obtention du titre de

Docteur en informatique

préparée dans le cadre d'une cotutelle entre

Université de Pau et des Pays de l'Adour

et

Université Constantine 2 - Abdelhamid Mehri

présentée et soutenue par

Khaled KHEBBEB

Le 29 Juin 2019 à Université Constantine 2

Composition du jury

Président:	Pr. Mahmoud BOUFAIDA	Université Constantine 2 - Abdelhamid Mehri, Algérie
Rapporteurs:	Pr. Yamine AIT-AMEUR	Institut National Polytechnique ENSEEIHT, Toulouse, France
	Pr. Kamel BARKAOUI	Conservatoire National des Arts et Métiers, Paris, France
Examinateurs:	Pr. Lakhdar DERDOURI	Université d'Oum El Bouaghi, Algérie
Directeurs:	Pr. Faiza BELALA	Université Constantine 2 - Abdelhamid Mehri, Algérie
	Dr. Nabil HAMEURLAIN	Université de Pau et des Pays de l'Adour, France

Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour (LIUPPA EA 3000)
Laboratoire d'Informatique Répartie (LIRE)

"Without judgment perception would increase a million times."

Chuck Schuldiner

Remerciements

Il me sera très difficile de remercier tout le monde car c'est grâce à l'aide de nombreuses personnes que j'ai pu mener cette thèse à son terme.

Je voudrais tout d'abord remercier grandement mes deux directeurs de thèse, madame Faiza Belala, professeur à l'université Constantine 2, et monsieur Nabil Hameurlain, maître de conférences habilité à diriger des recherches à l'université de Pau et des Pays de l'Adour (UPPA), pour toute leur aide. Je suis ravi d'avoir travaillé en leur compagnie car outre leur appui scientifique, leur pédagogie et leur patience, ils ont toujours été là pour me soutenir et me conseiller au cours de l'élaboration de cette thèse. J'exprime ma gratitude à Monsieur Hameurlain qui m'a dignement accueilli pendant trois années au sein de l'équipe MOVIES du laboratoire LIUPPA. C'est à ses côtés que j'ai compris ce que rigueur et précision voulaient dire.

Je tiens à remercier monsieur Kamel Barkaoui, professeur au centre national des arts et métiers (CNAM) de Paris, et monsieur Yamine Ait-Ameur, professeur à l'institut national polytechnique de Toulouse (INP ENSEEIHT), pour l'honneur qu'ils m'ont fait en acceptant d'être rapporteurs de cette thèse, et pour le temps qu'ils ont consacré à mon travail.

Je remercie monsieur Mauro Gaio et monsieur Ernesto Exposito, professeurs à l'université de Pau et des pays de l'Adour, et je réitère mes remerciements à monsieur Ait-Ameur, pour leur participation et leur disponibilité pendant les divers comités de suivi de thèse. Il ont pris le temps de m'écouter et de discuter avec moi. Leur remarques m'ont permis de faire avancer mon projet et d'envisager mon travail sous un autre angle.

Je tiens à remercier monsieur Mahmoud Boufaida, professeur à l'université Constantine 2 pour avoir accepté de présider mon jury de thèse, ainsi que monsieur Lakhdar Derdouri, professeur à l'université de Oum Bouaghi, pour avoir accepté d'être dans mon jury de thèse en tant qu'examinateur. Je les remercie pour l'honneur qu'ils me font, pour leur participation scientifique ainsi que pour le temps qu'ils ont consacré à ma recherche.

Un grand merci aux membres du département d'informatique du collège STEE de l'UPPA, aux membres du LIUPPA du site de Pau, ainsi qu'au personnel de l'école doctorale SEA (ED 211) de l'UPPA pour leur accueil et leur support, tout au long de la réalisation de mes travaux.

Je tiens particulièrement à remercier les personnes qui ont partagé mon quotidien durant ces années de thèse. Je pense aux amitiés tissées et à toutes ces personnes qui ont su être présentes en toutes circonstances.

Mes derniers remerciements vont à ma famille, et en particulier à mon père, qui a tout fait pour m'aider, qui m'a soutenu et supporté dans tout ce que j'ai entrepris.

Résumé

L'élasticité est une propriété qui permet aux systèmes Cloud de s'auto-adapter à leur charge de travail en provisionnant et en libérant des ressources informatiques, de manière autonomique, lorsque la demande augmente et diminue. En raison de la nature imprévisible de la charge de travail et des nombreux facteurs déterminant l'élasticité, fournir des plans d'action précis pour gérer l'élasticité d'un système cloud, tout en respectant des politiques de haut niveau (performances, cout, etc.) est une tâche particulièrement difficile. Les travaux de cette thèse visent à proposer, en utilisant le formalisme des bigraphes comme modèle formel, une spécification et une implémentation des systèmes Cloud Computing élastiques sur deux aspects : structurel et comportemental.

Du point de vue structurel, le but est de définir et de modéliser une structure correcte des systèmes Cloud du côté " backend ". Cette partie est supportée par les capacités de spécification fournies par le formalisme des Bigraphes, à savoir : le principe de "sorting" et de règles de construction permettant de définir les desiderata du concepteur. Concernant l'aspect comportemental, il s'agit de modéliser, valider et implémenter des stratégies génériques de mise à l'échelle automatique en vue de décrire les différents mécanismes d'auto-adaptation élastiques des systèmes cloud (mise à l'échelle horizontale, verticale, migration, etc.), en multi-couches (i.e., aux niveaux service et infrastructure). Ces tâches sont prises en charge par les aspects dynamiques propres aux Systèmes Réactifs Bigraphiques (BRS) notamment par le biais des règles de réaction.

Les stratégies d'élasticité introduites visent à guider le déclenchement conditionnel des différentes règles de réaction définies, afin de décrire les comportements d'auto-adaptation des systèmes Cloud au niveau service et infrastructure. L'encodage de ces spécifications et leurs implémentations sont définis en logique de réécriture via le langage Maude. Leur bon fonctionnement est vérifié formellement à travers une technique de model-checking supportée par la logique temporelle linéaire LTL.

Afin de valider ces contributions d'un point de vue quantitatif, nous proposons une approche à base de file d'attente pour analyser, évaluer et discuter les stratégies d'élasticité d'un système Cloud à travers différents scénarios simulés. Dans nos travaux, nous explorons la définition d'une "bonne" stratégie en prenant en compte une étude de cas qui repose sur la nature changeante de la charge de travail. Nous proposons une manière originale de composer plusieurs stratégies d'élasticité à plusieurs niveaux afin de garantir différentes politiques de haut-niveau.

Mots-clé: Cloud Computing, Systèmes auto-adaptatifs, Modélisation et vérification formelles, Systèmes Réactifs Bigraphiques, Logique de réécriture, Théorie des files d'attente.

Abstract

Elasticity property allows Cloud systems to adapt to their incoming workload by provisioning and de-provisioning computing resources in an autonomic manner, as the demand rises and drops. Due to the unpredictable nature of the workload and the numerous factors that impact elasticity, providing accurate action plans to insure a Cloud system's elasticity while preserving high level policies (performance, costs, etc.) is a particularly challenging task. This thesis aims at providing a thorough specification and implementation of Cloud systems, by relying on bigraphs as a formal model, over two aspects: structural and behavioral.

Structurally, the goal is to define a correct modeling of Cloud systems' "back-end" structure. This part is supported by the specification capabilities of Bigraph formalism. Specifically, via "sorting" mechanisms and construction rules that allow defining the designer's desiderata. As for the behavioral part, it consists of model, implement and validate generic elasticity strategies in order to describe Cloud systems' auto-adaptive behaviors (i.e., horizontal and vertical scaling, migration, etc.) in a cross-layer manner (i.e., at service and infrastructure levels). These tasks are supported by the dynamic aspects of Bigraphical Reactive Systems (BRS) formalism (through reaction rules).

The introduced elasticity strategies aim at guiding the conditional triggering of the defined reaction rules, in order to describe Cloud systems' auto-scaling behaviors in a cross-layered manner. The encoding of these specifications and their implementation are defined in Rewrite Logic via Maude language. Their correctness is formally verified through a model-checking technique supported by the linear temporal logic LTL.

In order to quantitatively validate these contributions, we propose a queuing-based approach in order to evaluate, analyze and discuss elasticity strategies in Cloud systems through different simulated execution scenarios. In this work, we explore the definition of a "good" strategy through a case study which considers the changing nature of the input workload. We propose an original way de compose different cross-layer elasticity strategies in order to guarantee different high-level policies.

Keywords: Cloud Computing, Self-adaptive systems, Formal modeling and verification, Bigraphical Reactive Systems, Rewrite logic, Queuing Theory.

Table des matières

Table des figures iv

Liste des tableaux vi

Liste des acronymes vii

1 Introduction générale 2
- 1.1 Contexte 2
- 1.2 Problématique 3
- 1.3 Objectifs et contributions 5
- 1.4 Plan de thèse 6
- 1.5 Diffusion scientifique 7

I État de l'Art 8

2 Principaux concepts et définitions 10
- 2.1 Introduction 10
- 2.2 Les systèmes Cloud Computing 11
  - 2.2.1 Définitions 11
  - 2.2.2 Caractéristiques 12
  - 2.2.3 Modèles de service 12
  - 2.2.4 Modèles de déploiement 14
  - 2.2.5 Acteurs du Cloud 15
  - 2.2.6 Virtualisation 16
  - 2.2.7 Modèle économique 17
  - 2.2.8 Efficience et Cloud Computing 19
- 2.3 Élasticité dans le cloud 20
  - 2.3.1 Définitions 20
  - 2.3.2 Niveaux d'action de l'élasticité 22
  - 2.3.3 Stratégies d'élasticité et techniques sous-jacentes 22
  - 2.3.4 Objectifs de l'élasticité 24
  - 2.3.5 Méthodes d'élasticité 25
- 2.4 Informatique autonomique 27
  - 2.4.1 Définitions 27
  - 2.4.2 Propriétés autonomiques 27
  - 2.4.3 Boucle de contrôle autonomique 29
  - 2.4.4 Contrôleur d'élasticité autonomique dans les systèmes Cloud 31
- 2.5 Conclusion 32

3 Travaux existants sur l'élasticité dans le Cloud 34
- 3.1 Environnements pour la gestion autonomique de l'élasticité dans le Cloud 34

3.1.1 Synthèse . . . 39
3.2 Modèles formels pour l'élasticité dans le Cloud . . . 40
3.2.1 Synthèse . . . 44
3.3 Conclusion . . . 46

4 Prérequis et fondements formels **48**
4.1 Introduction . . . 48
4.2 Les Systèmes Réactifs Bigraphiques . . . 49
4.2.1 Anatomie et forme graphique des bigraphes . . . 50
4.2.2 Définitions Formelles . . . 52
4.2.3 Opérations sur les bigraphes . . . 53
4.2.4 Forme algébrique . . . 56
4.2.5 Logique de typage (*sorting*) . . . 59
4.2.6 Dynamique des bigraphes . . . 60
4.2.7 Bigraphes concrets et bigraphes abstraits . . . 61
4.2.8 Outils pratiques autour des BRS . . . 62
4.3 Langage Maude . . . 63
4.3.1 Syntaxe et notations . . . 63
4.3.2 Modules fonctionnels . . . 64
4.3.3 Modules systèmes . . . 65
4.3.4 Vérification formelle dans Maude . . . 67
4.4 Théorie des files d'attente . . . 68
4.4.1 Objectif de l'analyse des files d'attente . . . 69
4.4.2 Caractéristiques des systèmes de files d'attente . . . 70
4.4.3 Modèles de files d'attente . . . 71
4.4.4 Théorie des files d'attente et gestion dynamique des ressources . . . 72
4.5 Conclusion . . . 73

II Contributions **74**

5 Méthodologie et contributions **76**
5.1 Contexte et objectifs . . . 76
5.2 Méthodologie et principes de la solution . . . 77
5.2.1 Modélisation formelle des systèmes Cloud élastiques . . . 77
5.2.2 Exécution et vérification de l'élasticité . . . 78
5.2.3 Évaluation de l'élasticité . . . 79
5.3 Organisation des contributions . . . 80

6 Formalisation de l'élasticité multi-couches dans le Cloud **82**
6.1 Introduction . . . 82
6.2 Approche bigraphique pour la modélisation des systèmes Cloud élastiques . . . 83
6.2.1 Méta-modèle d'un système Cloud élastique . . . 84
6.2.2 Sémantique bigraphique pour la structure d'un système Cloud . . . 85
6.2.3 Sémantique basée BRS pour la dynamique d'un système Cloud élastique . 88
6.2.4 Bilan et problématique . . . 93
6.3 Stratégies d'élasticité multi-couches dans le Cloud . . . 94
6.3.1 Dimensionnement Horizontal . . . 97
6.3.2 Migration et Load Balancing . . . 99
6.3.3 Dimensionnement Vertical . . . 100
6.3.4 Bilan . . . 102
6.4 Conclusion . . . 102

7 Implémentation et vérification du comportement élastique d'un système Cloud 104
7.1 Introduction 104
7.2 Encodage des spécifications bigraphiques dans Maude 105
7.2.1 Principes de l'encodage 105
7.2.2 Encodage des aspects structurels (module fonctionnel) 106
7.2.3 Encodage des stratégies d'élasticité (module système) 108
7.2.4 Bilan 110
7.3 Vérification du comportement élastique 111
7.3.1 Comportement et propriétés élastiques 111
7.3.2 Encodage d'états et de propriétés dans Maude 114
7.3.3 Vérification de propriétés 115
7.3.4 Bilan 118
7.4 Conclusion 118

8 Évaluation quantitative des stratégies d'élasticité multi-couches 120
8.1 Introduction 120
8.2 Étude de cas : modélisation de la plateforme Steam ® 121
8.3 Évaluation outillée de l'élasticité à base de files d'attente 124
8.3.1 Modèle de files d'attente 124
8.3.2 Principes de la simulation à base de files d'attente 126
8.3.3 Un outil pour la simulation de l'élasticité dans le Cloud 129
8.4 Simulation et évaluation des comportements élastiques de la plateforme Steam . 131
8.4.1 Paramétrage des simulations 131
8.4.2 Stratégies de dimensionnement Horizontal 133
8.4.3 Migration et Load Balancing 140
8.4.4 Stratégies de dimensionnement Vertical 140
8.4.5 Comparaison de l'élasticité Horizontale et Verticale 143
8.5 Conclusion et discussion 144

9 Conclusion Générale 148
9.1 Contributions 149
9.2 Perspectives 150

Bibliographie 154

Table des figures

2.1 Le Cloud Computing selon le NIST [Dupont, 2016] . . . 13
2.2 Répartition des tâches d'administration des modèles de service . . . 14
2.3 Modèles de déploiement [Calliope, 2016] . . . 15
2.4 Infrastructure virtualisée [Dupont, 2016] . . . 16
2.5 Architectures des machines virtuelles et des conteneurs Docker . . . 17
2.6 Illustration de la tarification horaire par type d'instance de VM par *Amazon* . . . 18
2.7 Illustration de la tarification horaire par type d'instance de VM par MS *Azure* . . . 18
2.8 Optimisation de l'utilisation des ressources : comparaison entre les modèles Cluster et Cloud. . . . 20
2.9 Quadruplet de l'élasticité [Galante and Bona, 2012] . . . 22
2.10 Élasticité : sur/sous-dimensionnement dans les modèles Cluster et Cloud . . . 25
2.11 Dimensionnement horizontal et vertical au niveau Infrastructure . . . 26
2.12 Boucle de contrôle autonomique MAPE-K . . . 30
2.13 Vue d'ensemble du comportement élastique autonomique d'un système Cloud . . . 31

3.1 Vulcan, un gestionnaire de planification de l'élasticité Vulcan [Letondeur, 2014] . . . 35
3.2 Coordination de contrôleurs d'élasticité entre VMs et conteneurs Docker [Al-Dhuraibi, 2018] . . . 36
3.3 Architecture générale du Framework SCUBA [Dupont, 2016] . . . 36
3.4 Architecture du Framework ElaaS [Kranas et al., 2012] . . . 37
3.5 Architecture du Framework Vadara [Loff and Garcia, 2014] . . . 38
3.6 Métriques de surveillance de l'élasticité de MELA [Moldovan et al., 2015] . . . 38
3.7 Architecture du Framework QoS-Aware Resource Elasticity (QRE) [Kaur and Chana, 2014] . . . 39
3.8 Principe de modélisation à base de files d'attente [Yataghene et al., 2014] . . . 42
3.9 Vue d'ensemble du contrôleur d'élasticité pour les SBP [Amziani, 2015] . . . 42
3.10 Fonctionnement de MoVeElastic [Sahli, 2017] . . . 43

4.1 Anatomie des bigraphes . . . 51
4.2 Graphe de places . . . 51
4.3 Graphe de liens . . . 51
4.4 Composition de deux graphes de places : $A^P = G^P \circ F^P$ . . . 54
4.5 Composition de deux graphes de liens : $A^L = G^L \circ F^L$ . . . 54
4.6 Composition de deux bigraphes : $A = G \circ F$ . . . 55
4.7 Produit tensoriel de deux graphes de places : $B^P = B_0^P \otimes B_1^P$ . . . 56
4.8 Produit tensoriel de deux graphes de liens : $B^L = B_0^L \otimes B_1^L$ . . . 56
4.9 Produit tensoriel de deux bigraphes : $B = B_0 \otimes B_1$ . . . 57
4.10 Bigraphe U modélisant le bâtiment B d'une université . . . 60
4.11 Exemple d'une règle de réaction bigraphique . . . 61
4.12 Analyse des files d'attente [Gueroui, 2015] . . . 70
4.13 Système de file d'attente simple [Gueroui, 2015] . . . 70

4.14 Nombre de serveurs dans un système de files d'attente 71
4.15 Répartition des requêtes dans un système distribué 72

5.1 Vue d'ensemble des contributions 77
5.2 Modélisation et validation 80

6.1 Meta modèle : vision d'ensemble d'un système Cloud élastique 84
6.2 Exemple d'un bigraphe CS modélisant un système Cloud 88
6.3 Modélisation des actions d'adaptations élastiques par des règles de réaction bigraphiques 89
6.4 Forme graphique des règles de réaction $R3$ et $R6$ 91
6.5 Forme graphique des règles de réaction $R7$ et $R10$ 92
6.6 Forme graphique des règles de réaction $R11$ et $R12$ 93
6.7 Forme graphique des règles de réaction $R13$ et $R14$ 94
6.8 Forme graphique de la règle de réaction $R15$ 96

7.1 Vue d'ensemble du principe d'encodage des spécifications bigraphiques dans Maude . 106
7.2 Système de transition pour l'élasticité horizontale 113
7.3 Système de transition pour l'élasticité verticale 113
7.4 Vue d'ensemble de la solution de spécification, d'exécution et de vérification dans Maude 115
7.5 Résultat de la vérification de l'élasticité horizontale sous LTL Maude 116
7.6 Contre-exemple résultant de la vérification de l'élasticité horizontale par la négation 117
7.7 Résultat de la vérification de l'élasticité verticale sous LTL Maude 117
7.8 Contre-exemple résultant de la vérification de l'élasticité horizontale par la négation 118

8.1 Représentation bigraphique d'une configuration du service de vente en ligne Steam . 122
8.2 Principe d'application de l'approche file d'attente sur les systèmes Cloud 125
8.3 Impact du taux de service et de la taille de la file d'attente 127
8.4 Modèle de workload imprévisible 128
8.5 Infrastructures statiques vs. infrastructures élastiques 129
8.6 Structure fonctionnelle de l'outil de simulation de l'élasticité 129
8.7 Monitoring du scénario H1 134
8.8 Monitoring du scénario H2 135
8.9 Monitoring du scénario H3 136
8.10 Monitoring du scénario H4 138
8.11 Évaluation de l'élasticité horizontale en multi-couches 139
8.12 Trace du monitoring de H2 à t=93 140
8.13 Monitoring de l'élasticité verticale (scénario V) 142
8.14 Comparaison de l'élasticité horizontale et verticale 144

Liste des tableaux

3.1 Étude d'environnements pour la gestion autonomique de l'élasticité dans le Cloud . 40
3.2 Étude de modèles formels pour la spécification et la vérification de l'élasticité dans le Cloud . . . 45

4.1 Principaux termes du langage algébrique des bigraphes . . . 58

6.1 Contrôles et sortes du bigraphe CS . . . 86
6.2 Règles de construction Φ_{CS} du bigraphe CS . . . 87
6.3 Règles de réaction pour le dimensionnement et la migration des ressources . . . 90
6.4 Règles de réaction pour l'étiquetage des états . . . 96
6.5 Stratégies d'élasticité Horizontale . . . 98
6.6 Stratégies de migration et de load balancing . . . 100
6.7 Stratégies de dimensionnement Vertical . . . 101

7.1 Principales déclarations du module fonctionnel *Elastic_Cloud_System* . . . 107
7.2 Principales déclarations du module système *Elastic_Cloud_Behavior* . . . 109
7.3 Principales déclarations du module *Elastic_Cloud_Properties* . . . 116

8.1 Résultats du scénario H1 . . . 134
8.2 Résultats du scénario H2 . . . 135
8.3 Résultats du scénario H3 . . . 137
8.4 Résultats du scénario H4 . . . 138
8.5 Résultats du scénario V . . . 142
8.6 Comparaison de l'élasticité horizontale et verticale . . . 144

Liste des acronymes

Amazon EC2	Amazon Elastic Compute Cloud
AP	Atomic Propositions
API	Application Programming Interface
BigMC	Bigraphical Model Checker
BigraphER	Bigraph Evaluator and Rewriting
BPL	Bigraphical Programming Language Project
BRS	Bigraphical Reactive Systems
CdM	Chaîne de Markov
CLTLt(d)	Timed Constraint Linear Temporal Logic
CPN	Colored Petri Nets
CPU	Central Processing Unit
ElaaS	Elasticity as a Service
FIFO	First In First Out
IaaS	Infrastructure as a Service
LB	Load Balancer
LTL	Logique Temporelle Linéaire
LTS	Labeled Transition System
MAPE-K	Monitor, Analyse, Plan, Execute - Knowledge
MFA	Modèle de Files d'Attente
MS	Microsoft
NIST	National Institute of Standards and Technology
OCaml	Objective Categorical Abstract Machine Language
OCCI	Open Cloud Computing Interface
OR	Objectif de Recherche
OS	Operating System
PaaS	Platform as a Service
PC	Personal Computer
PM	Machine Physique
PN	Petri Nets
PUE	Power Usage Efficiency
QoE	Quality of Experience
QoS	Quality of Service
QRE	QoS-aware resource elasticity
RAM	Random Access Memory
SaaS	Software as a Service
SAT	Satisfiability
SBP	Service-based Businesss Process
SJF	Shortest Job First

SLA Service Level Agreement
SMT Satisfiability modulo theories
SRB Systèmes Réactifs Bigraphiques
TFA Théorie des Files d'Attente
TIC Technologies de l'Information et de la Communication
VM Machine Virtuelle

Chapitre 1

Introduction générale

Sommaire

1.1 Contexte 2
1.2 Problématique 3
1.3 Objectifs et contributions 5
1.4 Plan de thèse 6
1.5 Diffusion scientifique 7

1.1 Contexte

Depuis leur émergence, les technologies de l'information et de la communication (TIC) ont connu une évolution constante suivant un rythme très soutenu. Afin de maintenir une présence sur le marché des TIC de plus en plus exigent, de nombreuses organisations, tant industrielles qu'académiques, ont œvré pour réduire les coûts de fonctionnement des systèmes informatiques, assurer leur mise à l'échelle, fournir de bonnes performances à leurs applications, tout en optimisant les coûts liés à l'utilisation des ressources informatiques. Plusieurs technologies sont apparues au fil des années telles que les systèmes distribués, le traitement parallèle, le Grid Computing, la virtualisation et d'autres, afin de relever le défi de l'efficacité de la gestion des ressources et le déploiement des applications [Zhang et al., 2010]. Cependant, avec la croissance de nouvelles exigences métier, notamment en termes de flexibilité, de souplesse, de simplicité, d'économies financières ou encore d'efficience énergétique, le paradigme Cloud Computing ou "informatique en nuage" a émergé cette dernière décennie et s'est imposé en apportant une ère de renouveau. En se basant sur ces différentes technologies, le Cloud apporte de nouvelles caractéristiques et principes à l'approvisionnement des ressources informatiques. La terme "nuage" est une métaphore qui désigne en réalité un réseau de ressources informatiques (matérielles et logicielles) qui se présentent sous la forme de services qu'un utilisateur peut consommer à la demande via un réseau, généralement internet, et selon un modèle de facturation "*pay-as-you-go*", proportionnel à l'utilisation réelle des ressources [Chan, 2014, Fox et al., 2009].

Le Cloud introduit de nombreux avantages tels que le déploiement plus facile et plus rapide des applications au sein d'infrastructures Cloud, avec l'assurance d'une disponibilité accrue et une fiabilité optimale [Suleiman et al., 2012]. Cela en fait l'un des modèles les plus adoptés dans le monde de l'industrie, de la recherche scientifique ainsi que par le grand public. Afin de satisfaire les exigences de ses usagers, le Cloud offre l'accès à des service différents selon les besoins spécifiques d'un client. Les services Cloud peuvent

être classifiés selon trois couches ou modèles : infrastructure en tant que service (IaaS), plateforme en tant que service (PaaS) et logiciel en tant que service (SaaS). Ces modèles présentent des avantages et des inconvénients. Précisément, la souplesse d'utilisation et la capacité de contrôle des services, ainsi que les connaissances requises pour les utiliser tendent à diminuer du niveau IaaS au niveau SaaS. Un service IaaS propose l'accès à des ressources virtualisées (e.g. Machines Virtuelles - VMs) que l'utilisateur doit configurer, ce qui n'est pas à la portée des usagers non spécialisés. D'un autre côté, un utilisateur d'un service SaaS peut interagir avec son application à travers une interface de programmation (API) facilement exploitable, mais n'a aucun contrôle sur l'infrastructure Cloud sous-jacente.

Avec l'évolution constante des exigence du marché des TIC et de celles de ses usagers, le Cloud incarne désormais un environnement hautement dynamique et variable à large échelle. Dans ce contexte, le paradigme Cloud se distingue des autres paradigmes par l'une de ses caractéristiques principales : l'élasticité. L'élasticité dans le Cloud est un concept qui vise à optimiser la gestion des ressources informatiques. L'idée est de permettre à un système Cloud, dit "élastique" ou doté d'un "comportement élastique" [Bersani et al., 2014], de répondre efficacement aux sollicitions variables de ses clients en termes d'utilisation de ressources informatiques. Cela consiste à rajouter et à retirer des ressources informatiques en vue de s'adapter aux changement dans sa charge de travail actuelle. L'enjeu consiste à maintenir une qualité de service optimale, tout en minimisant les coûts liés à l'utilisation des ressources informatiques. L'élasticité fournit des mécanismes qui permettent à un système Cloud de supporter, au mieux, une montée de la charge de travail puis un retour à la normale, sans interruption de service et si possible sans répercussions sur la qualité du service. D'un autre côté, l'élasticité permet de respecter les engagements "*pay-as-you-go*" en fournissant des mécanismes de mise à l'échelle en cas de baisse de la charge, dans le but d'éviter les sur-facturations inutiles. En d'autres termes, l'élasticité incarne l'un des rouages nécessaires à l'optimisation des ressources (i.e., optimisation de coûts), tout en assurant aux utilisateurs un niveau de service optimal (i.e., optimisation des performances), dans un contexte de plus en plus dynamique et variable [Herbst et al., 2013].

Un système Cloud peut adopter un comportement élastique au niveau de l'une ou plusieurs de ses couches (IaaS, PaaS et SaaS). Lorsque l'élasticité est appliquée sur plus d'une couche du Cloud, on parle alors d'élasticité multi-couches ou encore de comportement élastique multi-couches [Kouki and Ledoux, 2013, Copil et al., 2013]. L'élasticité est assurée selon des stratégies réactives ou prédictives et via trois méthodes : le dimensionnement horizontal (ou élasticité horizontale), le dimensionnement vertical (ou élasticité verticale) et la migration. L'élasticité horizontale consiste à répliquer ou supprimer des instances de VMs ou de services (couche IaaS ou SaaS respectivement). L'élasticité verticale augmente ou réduit la quantité de ressources allouées à une VM ou un conteneur (selon la couche IaaS ou PaaS respectivement). Enfin, la méthode de migration consiste à déplacer ou redéployer une instance d'un hôte vers un autre hôte.

1.2 Problématique

En vue des exigences métier liées à la gestion des ressources informatiques dans le Cloud telles que la fiabilité, la réactivité et l'efficacité, l'élasticité est généralement gérée de manière autonomique. Elle est assurée de manière automatisée et en minimisant les interventions humaines dont l'efficacité décline, face à ces défis de plus en plus complexes.

Ainsi, l'élasticité est généralement assurée par un contrôleur d'élasticité : une entité autonomique qui régit le comportement élastique d'un système Cloud, lui conférant des capacités d'auto adaptation en termes de gestion dynamique des ressources [Jamshidi et al., 2014].

Plusieurs grands acteurs de l'industrie du Cloud proposent des solutions pour la gestion autonomique de l'élasticité tels que Rackspace, Amazon EC2, Google AppEngine et Microsoft Azure. Ces solutions commerciales ne sont globalement pas totalement matures, elles sont relativement récentes et présentent des limites en termes de contrôle d'élasticité et d'adaptation dynamique de la consommation de ressources. Cela influe sur la disponibilité, la performance ainsi que sur la facturation des services utilisés [Gambi et al., 2016]. De plus, les mécanismes proposés par ces solutions industrielles se limitent souvent à la spécification des services Cloud requis, sans tenir compte de leurs comportements élastiques. D'un autre côté, plusieurs solutions issues du monde académique mettent au point des contrôleurs d'élasticité visant à gérer, de manière autonomique, l'allocation des ressources dans le Cloud. Ces solutions sont souvent basés sur un principe de boucles de contrôle autonomique fermées, généralement selon le modèle MAPE-K (Monitor, Analyse, Plan, Execute - Knowledge) introduit par IBM [Jacob et al., 2004]. Selon ce modèle, le contrôleur d'élasticité surveille le système Cloud géré afin de recueillir des informations sur son état en termes de performances, de consommation de ressources, etc. [Trihinas et al., 2014]. Ensuite, ces informations sont analysées afin de diagnostiquer d'éventuels états non-désirés. Enfin, le contrôleur d'élasticité décide des actions à déclencher, en termes de méthodes d'élasticité, et établit un plan d'action à exécuter afin de corriger, si besoin, l'état du système Cloud géré.

Les solutions existantes ne décrivent globalement pas de comportements génériques et traitent d'un aspect particulier de l'élasticité. Précisément, certaines se focalisent sur l'élasticité horizontale et la migration tandis que d'autres traitent uniquement de l'élasticité verticale. Certaines se concentrent sur une ou plusieurs couches du Cloud (IaaS, PaaS, SaaS) et d'autres introduisent des stratégies d'élasticité uniquement à un seul niveau. Certaines solutions introduisent un modèle d'élasticité réactive tandis que d'autres proposent un modèle d'élasticité prédictive. Enfin, la plupart des solutions proposées dans la littérature sont coûteuses et très difficiles à mettre en œuvre dans les environnements Cloud. En outre, elles peuvent provoquer des comportements indésirables, si elles ne sont pas correctement conçues.

Le comportement élastique d'un système Cloud dépend de la combinaison d'une multitude de facteurs tels que la quantité de ressources disponible, la charge de travail en entrée, la logique gouvernant le comportement du contrôleur d'élasticité ou encore les différentes politiques de haut niveau et propriétés à satisfaire. La complexité de ces dépendances, ainsi que la maitrise des potentiels effets de bords néfastes sur l'état global du système, rendent la conception des systèmes Cloud élastiques très difficile.

Face à la complexité grandissante des systèmes Cloud, leur conception est devenue de plus en plus difficile à maîtriser, et leur maintenance, de plus en plus coûteuse à assurer. Les méthodes de développement classiques, bien que coûteuses, se montrent souvent inefficaces pour garantir de manière exhaustive le champ de validité du comportement élastique d'un système Cloud, et peinent à assurer sa fiabilité et robustesse.

1.3 Objectifs et contributions

Afin de fournir une bonne gestion de l'élasticité, il est indispensable de s'appuyer sur un modèle qui permet de décrire les architectures des systèmes Cloud, ainsi que leur comportement élastique. La modélisation est une tâche impérative qui permet de déterminer et comprendre les changements structurels et les dépendances comportementales dans un système Cloud élastique afin d'éviter l'émergence des comportements provoquant des situations indésirables telles que l'instabilité dans l'allocation des ressources et la dégradation de la qualité de service. Dans ce contexte, les méthodes formelles caractérisées par leur efficacité, fiabilité et précision, présentent une solution efficace pour relever ce défi et pour éliminer les ambiguïtés sémantiques et la complexité provenant de la tâche de modélisation. Ceci permet de raisonner rigoureusement sur les spécifications formelles obtenues pour démontrer leur validité à travers plusieurs techniques de simulations et vérifications formelles. De plus, afin de garantir l'efficacité des comportements élastiques définis, il est primordial de les évaluer et de les valider du point de vue quantitatif, avant de les utiliser dans des environnement Cloud réels.

Afin de proposer une solution complète pour la gestion et l'analyse de l'élasticité dans les systèmes Cloud, nous identifions les principaux objectifs de recherche (OR) suivants :

OR1- Définir les comportements élastiques multi-couches d'un système Cloud.

OR2- Assurer une exécution autonomique de ces comportements.

OR3- Vérifier le bon fonctionnement de ces comportements.

OR4- Évaluer les performances et les coûts liés à ces comportements.

En d'autres termes, l'objectif général de cette thèse tend à proposer une solution à fondements formels pour la spécification des systèmes Cloud élastiques, la vérification du bon fonctionnement de leurs comportements élastiques ainsi que l'évaluation de leurs performances. Le présent manuscrit présente les contributions suivantes :

1. *Sémantique bigraphique structurelle :* Dans un premier temps, il s'agit de proposer un cadre formel la spécification des aspects structurels des systèmes Cloud élastiques. Un tel modèle permettrait d'exprimer les éléments pertinents d'une architecture Cloud en multi-couche, c'est à dire aux niveaux infrastructure et application. Nous proposons un modèle basé sur le formalisme des bigraphes et leur logique de typage associée afin décrire une sémantique robuste et détaillées pour les systèmes Cloud et des entités les composant.

2. *Sémantique BRS comportementale et stratégies d'élasticité multi-couches :* En complément de la précédente contribution, nous définissons une sémantique basée sur les systèmes réactifs bigraphiques (BRS) pour la modélisation des actions de reconfiguration dynamiques des systèmes Cloud. Les reconfigurations du système sont modélisées en termes de règles de réaction bigraphiques s'appliquant au niveau des différentes ressources matérielles et logicielles déployées au sein du système (multi-couches). À partir des différentes actions identifiées, l'idée et de proposer plusieurs stratégies pour le contrôle des reconfigurations du système. Précisément, nous avons introduit des stratégies décrivant les différentes méthodes d'élasticité

(horizontale, verticale, migration et load balancing) pouvant s'appliquer en multi-couche, à différents niveaux du système (infrastructure, application, ressources).

3. *Représentation de l'état élastique des systèmes Cloud :* Nous proposons une solution pour la représentation et l'expression des états du point de vue de l'élasticité d'un système Cloud. Les états identifiés servent à dégager des états désirables et indésirables en termes d'élasticité, servant à guider le dimensionnement élastique d'un système Cloud.

4. *Implémentation, exécution autonomique et vérification formelle des stratégies d'élasticité :* Nous présentons un encodage des spécifications bigraphiques et des stratégies d'élas-ticité dans le langage de spécification formelle Maude. Cet encodage permet de préserver la sémantique structurelle des systèmes Cloud tout en l'enrichissant d'aspects quantitatifs et la possibilité d'exprimer les états du système à travers des prédicats de la logique du premier ordre. En outre, nous procédons à la vérification formelle de l'élasticité des systèmes Cloud modélisés. Cette vérification se base sur une technique de model-checking à base d'états, supportée par la logique temporelle linéaire LTL.

5. *Évaluation et validation quantitative :* Enfin, nous proposons une étude expérimentale des comportements élastiques introduits d'un système Cloud. Nous tentons d'évaluer ces comportements d'un point de vue quantitatif, à travers une étude de cas d'un système Cloud existant. Nous présentons une solution outillée, à base de files d'attente, afin de simuler le fonctionnement des différentes stratégies d'élasticité définies.

1.4 Plan de thèse

Ce manuscrit de thèse est découpé en deux grandes parties organisées comme suit :

1. La première partie est constituée de l'art de l'art de la thèse. Le Chapitre 2 traite du contexte dans lequel s'inscrit le sujet de la thèse au travers d'un certain de nombre de définitions et concepts de base liés au paradigme Cloud Computing. Le Chapitre 3 expose un état de l'art visant à recenser et analyser plusieurs travaux existants liés à l'élasticité dans le Cloud. Le but est d'identifier les manques et limites des solutions industrielles et académiques, en vue d'identifier les manques et limitations et ainsi dégager les enjeux de cette thèse. Enfin, le Chapitre 4 présente les principaux modèles et théories formelles, au centre de la thèse, afin de préparer le lecteur à une bonne compréhension des contributions qui y sont présentées.

2. La seconde partie présente les contributions scientifiques de cette thèse. Elle est introduite par le Chapitre 5 donnant une vision d'ensemble des contributions. Le Chapitre 6 présente notre approche à base de systèmes réactifs bigraphiques pour la gestion autonomique de l'élasticité multi-couches dans le Cloud, en couvrant les aspects structurels et comportementaux et en introduisant des stratégies pour le dimensionnement élastique (horizontal, vertical, load balancing et migration) en multi-couches (infrastructure et application) d'un systtème Cloud. Le Chapitre 7 présente une approche pour l'implémentation, l'exécution et la vérification des comportements élastiques d'un système Cloud. Nous y présentons un encodage, dans

Maude, des spécifications bigraphiques introduites ainsi qu'une solution pour la vérification formelle des comportements élastiques à travers une technique de model-checking supporté par la logique LTL. Enfin, le Chapitre 8 présente notre approche outillée pour la simulation et l'évaluation de l'élasticité d'un système Cloud. Nous y appliquons notre approche de modélisation formelle sur une étude de cas d'un système Cloud existant, puis nous simulons, analysons et évaluons ses comportements élastiques.

1.5 Diffusion scientifique

Les travaux présentés dans ce manuscrit ont fait l'objet de plusieurs publications listées ci-dessous.

Revue scientifique avec comité de lecture

— Khebbeb, K., Hameurlain, N., Belala, F., Sahli, H. (2018) Formal Modelling and Verifying Elasticity Strategies in Cloud Systems. In : IET Software. 13(1), pp. 25-35. The Institution of Engineering and Technology [Khebbeb et al., 2018b].

Conférence internationale avec comité de lecture

— Khebbeb K., Hameurlain N., Belala F. (2018) Modeling and Evaluating Cross-layer Elasticity Strategies in Cloud Systems. In : Abdelwahed E., Bellatreche L., Golfarelli M., Méry D., Ordonez C. (eds) Model and Data Engineering. MEDI 2018. Lecture Notes in Computer Science, vol 11163. pp. 168-183. Springer, Cham [Khebbeb et al., 2018a].

— Khebbeb K., Sahli H., Hameurlain N., Belala F. (2017) A BRS Based Approach for Modeling Elastic Cloud Systems. In : Braubach L. et al. (eds) Service-Oriented Computing – ICSOC 2017 Workshops. ICSOC 2017. Lecture Notes in Computer Science, vol 10797. pp. 5-17. Springer, Cham [Khebbeb et al., 2017].

Première partie

État de l'Art

Chapitre 2

Principaux concepts et définitions

Sommaire

2.1 Introduction 10
2.2 Les systèmes Cloud Computing 11
2.2.1 Définitions 11
2.2.2 Caractéristiques 12
2.2.3 Modèles de service 12
2.2.4 Modèles de déploiement 14
2.2.5 Acteurs du Cloud 15
2.2.6 Virtualisation 16
2.2.7 Modèle économique 17
2.2.8 Efficience et Cloud Computing 19
2.3 Élasticité dans le cloud 20
2.3.1 Définitions 20
2.3.2 Niveaux d'action de l'élasticité 22
2.3.3 Stratégies d'élasticité et techniques sous-jacentes 22
2.3.4 Objectifs de l'élasticité 24
2.3.5 Méthodes d'élasticité 25
2.4 Informatique autonomique 27
2.4.1 Définitions 27
2.4.2 Propriétés autonomiques 27
2.4.3 Boucle de contrôle autonomique 29
2.4.4 Contrôleur d'élasticité autonomique dans les systèmes Cloud 31
2.5 Conclusion 32

2.1 Introduction

Depuis leur émergence, les technologies de l'information et de la communication (TIC) ont connu une évolution constante, suivant un rythme très soutenu. Afin de maintenir une présence sur le marché des TIC de plus en plus exigent, de nombreuses organisations, tant industrielles qu'académiques, ont œvré pour trouver la meilleure façon de réduire les coûts de fonctionnement, assurer la mise à l'échelle de leurs systèmes informatiques, fournir une bonne performance à leurs applications tout en optimisant les coûts liés à l'utilisation des ressources informatiques. Plusieurs technologies sont apparues au fil des années, telles que les systèmes distribués, le traitement parallèle, le Grid Computing, la virtualisation et d'autres, traitant de l'efficacité de la gestion des ressources et le déploiement des

applications. Avec la croissance de nouvelles exigences métiers, notamment en termes de flexibilité, de souplesse, de simplicité, d'économies financières ou encore d'efficience énergétiques, le paradigme "Cloud Computing" a émergé cette dernière décennie, apportant une ère de renouveau. Ce paradigme se base sur ces technologies tout en apportant de nouvelles caractéristiques et principes à l'approvisionnement des ressources informatiques.

Dans ce premier Chapitre consacré à l'état de l'art, nous présentons le contexte dans lequel s'inscrit le sujet du présent manuscrit de thèse. Nous y présentons un certain nombre de définition de technologies et de concepts de base liés à nos travaux. Dans un premier temps, nous présentons, dans la Section 2.2 le paradigme Cloud Computing, ses définitions et ses caractéristiques. Ensuite, nous introduisons la notion d'élasticité, une caractéristique essentielle du Cloud. Nous aborderons sa définition, ses mécanismes et ses enjeux dans la gestion de l'allocation dynamique des ressources dans le cloud (Section 2.3). Enfin, nous discutons le contexte hautement dynamique et complexe qu'incarne le Cloud Computing dans la Section 2.4. Nous y présentons le domaine de l'informatique autonomique, une philosophie visant à doter les systèmes de capacités d'auto-administration.

2.2 Les systèmes Cloud Computing

Le Cloud Computing ou informatique en nuage est un paradigme récent de la technologie de l'information qui se base sur plusieurs technologies existantes. La terme « nuage » est une métaphore qui désigne en réalité un réseau de ressources informatiques (matérielles et logicielles) fournies sous forme de services que l'utilisateur peut consommer à la demande via un réseau, généralement internet, selon un modèle de facturation proportionnel à l'utilisation réelle des ressources [Chan, 2014, Fox et al., 2009, Dupont, 2016]. Le terme « Cloud Computing » est apparu vers la fin des années 2000. Ce paradigme est le résultat de la recherche et l'ingénierie en informatique visant à répondre aux problématiques liées au partage de ressources informatiques et à l'optimisation de leur utilisation. En effet, étant donnés les coûts élevés d'acquisition de matériel informatique (notamment dans les années 1960) et son faible taux d'utilisation, l'idée de fournir un service informatique publique centralisé, partagé et facilement accessible via un réseau a naturellement vu le jour et fut connue sous le nom d'informatique utilitaire [Garfinkel, 1999, Kleinrock, 1976]. À cette époque, cette vision fut utopique en vue des limitations technologiques. Par la suite, la maturation constante des technologies matérielles, logicielles et des réseaux aboutirent à une première concrétisation de ces objectifs (dans les années 1990) avec l'apparition du Grid Computing. Plus tard, les années 2000 connurent enfin l'apparition du paradigme Cloud Computing.

Dans cette Section, nous aborderons les différents aspects et concepts clés qui définissent le Cloud Computing. Nous évoquerons l'évolution de sa définition, ses caractéristiques, ses modèles de service et de déploiement ainsi que les différents acteurs au centre de son fonctionnement.

2.2.1 Définitions

Tout au long de sa maturation, le paradigme Cloud a connu de nombreuses définitions incomplètes et parfois approximatives, généralement se focalisant sur un aspect particulier de ce modèle [Buyya et al., 2008, Fox et al., 2009, Wang et al., 2010, Borenstein and Blake, 2011]. La définition la plus adoptée et largement acceptée comme la plus complète

a été apportée par le NIST en 2011 : *"Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models."* [Mell et al., 2011]. Selon le NIST, le modèle du Cloud Computing repose sur cinq caractéristiques principales, trois modèles de service et quatre modèles de déploiement (voir Figure 2.1).

2.2.2 Caractéristiques

Le cloud se distingue par les cinq caractéristiques essentielles suivantes :

— *Libre-service à la demande (on-demand self-service)* : L'utilisateur peut réserver ou libérer les ressources (CPU, stockage, bande pasasnte, etc.) en fonction de ses besoins sans interaction avec le fournisseur du service. Les ressources sont fournies d'une manière entièrement automatisée au client. Ce dernier peut gérer à distance ses ressources, généralement au moyen d'une interface.

— *Accès ubiquitaire via le réseau (broad network access)* : L'ensemble des ressources est accessible de partout (navigateurs WEB, smartphones, tablettes, etc.) via le réseau (Internet ou privé) en s'appuyant sur des mécanismes standards facilitant l'accès au service pour différents types de clients.

— *Mise en commun des ressources (resource pooling)* : Le fournisseur mutualise les ressources (ou services) qu'il attribue dynamiquement aux différents clients en fonction de la demande. Ce partage des ressources est la caractéristique qui différencie le Cloud Computing de autres modèles dits classiques.

— *Service mesuré (measured service)* : L'utilisateur est facturé en fonction de son utilisation en ressources ou services (selon la politique *pay-as-you-go*). Cette utilisation est contrôlée, mesurée et communiquée aux usagers ou fournisseur de service de façon transparente.

— *Élasticité rapide (rapid elasticity)* : L'accès aux ressources est assuré de manière souple et rapide. Les ressources peuvent être approvisionnées ou libérées rapidement, généralement de manière automatisée, pour répondre à des besoins qui évoluent vite (e.g., montée ou baisse soudaine de la charge de travail). Cette automatisation de l'approvisionnement des ressources est assurée de manière dite « autonomique » et est généralement assurée par un composant autonomique connu sous le nom de contrôleur d'élasticité (*elasticity controller*) [Bersani et al., 2014].

2.2.3 Modèles de service

Selon le NIST, le paradigme Cloud se manifeste en trois couches principales :

— *SaaS (Software as a Service)* : La couche des "logiciels en tant que service" regroupe les applications accessibles via internet où le matériel, l'hébergement, l'environnement d'application et le logiciel sont dématérialisés et dont la gestion est en dehors de la responsabilité de l'usager. Les applications de type SaaS sont diverses et variées. On peut citer les applications telles que la messagerie *Gmail*, l'éditeur de documents Office 365, le réseau social *Facebook*, les services de stockage de données *Google Drive* et *DropBox*, ou encore le service de cartographie en ligne *Google Maps*.

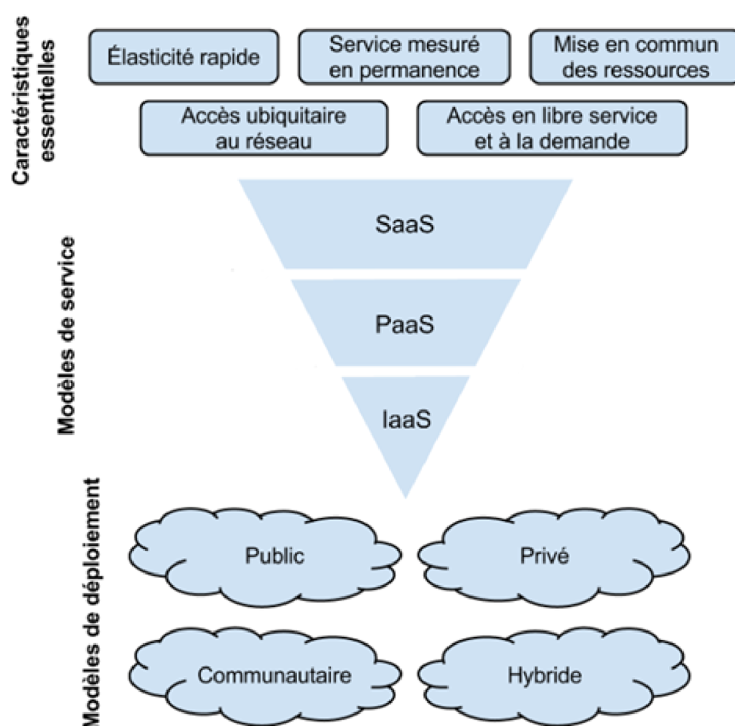


FIGURE 2.1 – Le Cloud Computing selon le NIST [Dupont, 2016]

— *IaaS (Infrastructure as a Service)* : Dans cette couche, la ressource fournie est une infrastructure informatique complète virtualisée. Physiquement, les ressources matérielles (*hardware*) proviennent d'une multitude de serveurs et de réseaux généralement distribués à travers de nombreux centres de données (datacenters). Une solution IaaS permet aux utilisateurs d'accéder de façon flexible à des systèmes virtuels complets en déployant /arrêtant à la demande des ressources virtuelles dans des *datacenters*, dont la responsabilité d'entretien (notamment du matériel sous-jacent) revient au fournisseur de de l'infrastructure Cloud. Parmi les acteurs principaux de IaaS, on retrouve *Amazon EC2* [Cloud, 2011], *Microsoft Azure* [Copeland et al., 2015] ou encore *RackSpace* [Rackspace, 2010]. Il existe aussi des solutions open-source telle que *OpenStack* [Sefraoui et al., 2012].

— *PaaS (Platform as a Service)* : La couche des "plateformes en tant que service" offre l'accès à un environnement de développement administré, hébergé et maintenu par le fournisseurs de la plateforme PaaS. Le but étant de faciliter le déploiement et l'exécution des applications SaaS en ajoutant une couche de services à la couche IaaS. En d'autres termes, le PaaS, situé entre la couche SaaS et la couche IaaS, abstrait la couche IaaS à ses utilisateurs. Cela permet aux équipes de développement de se concentrer sur l'architecture et la réalisation des applications sans se soucier des détails de configuration de l'infrastructure (matériel, réseau, etc.). Comme exemples de PaaS, on peut citer *Cloud Foundry*, *Google App Engine*, ainsi que le projet open-source *OpenShift* [OpenShift, 2019].

la Figure 2.2 montre les différentes responsabilités à la charge des fournisseurs des différents modèles de service, en détaillant les tâches d'administration et de gestion qui les incombent. Le modèle interne représente les modèles dits classiques où tous les détails de l'installation du système sont à la charge du fournisseur de service. Récemment, l'acro-

nyme *XaaS* (*Anything as a Service* [Schaffer, 2009]) ou « tout en tant que service » a vu le jour du fait du nombre croissant d'applications basées sur le concept d'externalisation de fonctionnalités sous forme de services (e.g. DaaS : Data as a Service, NaaS : Network as a Service, GaaS : Gaming as a Service, etc.).

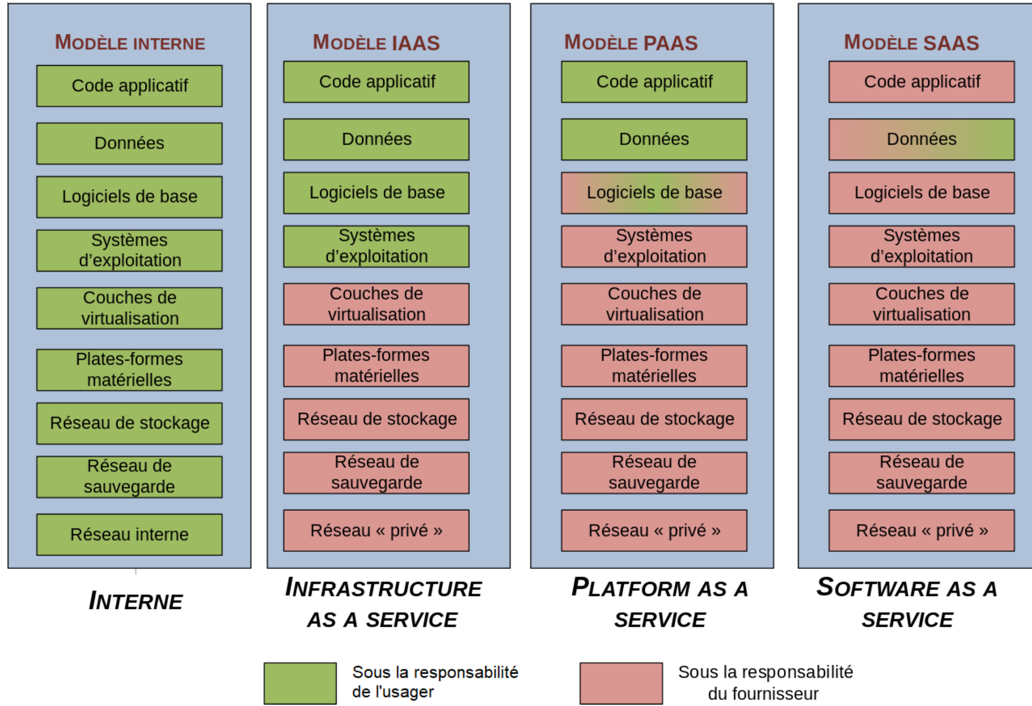


FIGURE 2.2 – Répartition des tâches d'administration des modèles de service

2.2.4 Modèles de déploiement

Le NIST définit quatre modèles de déploiement pour le Cloud Computing (voir Figure 2.3) :

— *Cloud privé* : il s'agit d'un environnement dont les ressources servent exclusivement l'entreprise utilisatrice. L'infrastructure peut être localisée/gérée sur place ou par un tiers mais l'entreprise est la seule à l'utiliser. Ce modèle offre un haut degré de contrôle et de sensibilité, permettant au propriétaire d'un service à facilement maitriser et administrer son système en termes de règlementations ou de sécurité, par exemple.

— *Cloud communautaire* : l'infrastructure est partagée entre plusieurs organisations et peut être gérée par le groupe ou par un tiers. Ce modèle est généralement adopté par les institutions gouvernementales, hôpitaux, hôtels, etc. afin de profiter de ressources communes (réseau, stockage, sécurité, etc.), ce qui assure un fonctionnement efficace des différents déploiements concernés.

— *Cloud public* : les ressources sont fournies par un prestataire, propriétaire de celles-ci, et mutualisées pour un usage partagé. Ce modèle est largement adopté par les fournisseurs de IaaS afin d'offrir une large offre de ressources partagées, à faible coût et gérées de manière élastique.

— *Cloud hybride* : il s'agit de la combinaison de plusieurs clouds indépendants publics ou privés. Ceux-ci doivent respecter des standards et des technologies communes pour assurer la portabilité des applications entre les clouds. Dans ce modèle de déploiement, une entreprise peut, par exemple, tirer parti du faible coût d'un cloud public pour l'hébergement de certains services classiques, tout en recourant à un cloud privé pour gérer des services plus sensibles (confidentialité, sécurité, etc.).

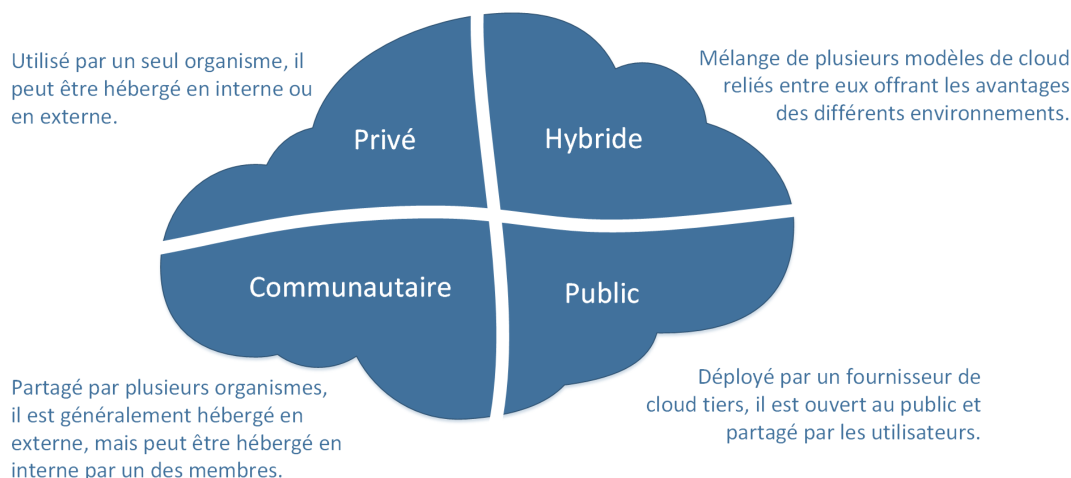


FIGURE 2.3 – Modèles de déploiement [Calliope, 2016]

2.2.5 Acteurs du Cloud

Selon le NIST, on distingue cinq acteurs majeurs au centre du fonctionnement du paradigme Cloud Computing : le fournisseur, l'usager, l'auditeur, le courtier et le transporteur de Cloud. Ces acteurs représentent une entité physique ou morale (une personne ou une organisation) qui prend part à un processus ou transactions dans le Cloud, de la manière suivante :

— *Le fournisseur (cloud provider)* : Représente une personne ou un organisme mettant un ensemble de services cloud à la disposition d'usagers (ou consommateurs) potentiels.

— L*'usager (cloud costumer)* : Représente une personne, un organisme ou une entité qui se prête à l'utilisation des différents services mis à disposition par le fournisseur, généralement via un contrat client.

— *L'auditeur (cloud auditor)* : Représente une entité dont la tâche est de mesurer et d'évaluer les services Cloud en termes de performances, sécurité, coûts, etc., tout en restant indépendant du fournisseur et des usagers du cloud.

— *Le courtier (cloud broker)* : Représente un parti qui s'occupe de la gestion et la prestation de service cloud en négociant les relations entre les usagers et le fournisseur Cloud.

— *Le transporteur (cloud carrier)* : Représente un organisme ou une entité intermédiaire qui s'occupe de fournir la connectivité et la disponibilité des services Cloud, en les transportant du fournisseur vers le consommateur.

2.2.6 Virtualisation

La virtualisation est une technique permettant de reproduire le comportement d'une machine physique (*Physical Machine* - PM) dans un environnement logiciel appelé machine virtuelle (*Virtual Machine* - VM). Cette technique donne à l'utilisateur l'illusion de manipuler une PM alors qu'en réalité, il interagit avec un environnement logiciel (VM). La virtualisation offre la possibilité d'exécuter plusieurs systèmes d'exploitation et/ou applications sur un seul serveur physique. Chaque VM exécute son propre système d'exploitation (*Operating System* - OS) permettant à l'utilisateur d'héberger des logiciels sur plusieurs plateformes différentes. Le système d'exploitation d'une VM est qualifié d'OS-*invité* pour le distinguer de celui de la PM, appelé OS-hôte. Le concept de virtualisation apporte de très nombreux avantages tels que l'optimisation de l'utilisation des ressources existantes par la mutualisation des ressources physiques qui résulte en une économie sur le matériel. Initialement, cette technique vise à « consolider » ou dématérialiser les infrastructures physiques afin de maximiser leur utilisation [Poniatowski, 2009]. Une VM est donc un conteneur de logiciels isolé, capable d'exécuter ses propres systèmes d'exploitation et applications. Les VMs sont créées et gérées par des logiciels appelés hyperviseurs (voir Figure 2.4) qui leur permet d'accéder directement au matériel de la PM. Les différentes ressources (CPU, RAM, stockage, réseau) sont allouées aux VMs de manière cloisonnée à partir d'une PM ce qui permet de maximiser leur utilisation. Ces ressources sont qualifiées de virtuelles mais sont en réalité bien réelles. [Smith and Nair, 2005].

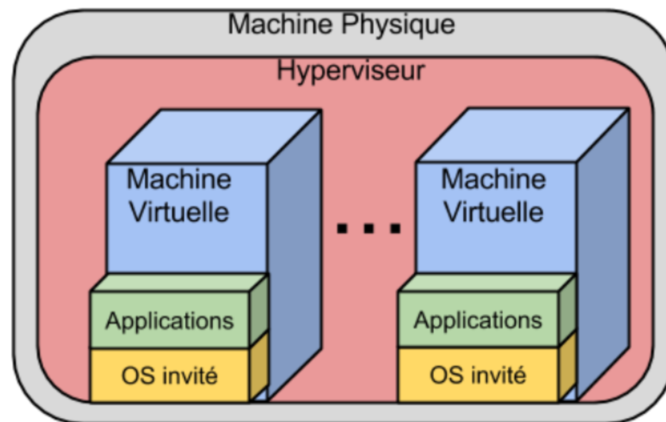


FIGURE 2.4 – Infrastructure virtualisée [Dupont, 2016]

La virtualisation complète apportée par les VMs implique l'embarquement du système d'exploitation en plus de l'ensemble des librairies et applications. Cette contrainte a amené la communauté à proposer une autre approche, notamment avec l'avènement de Docker [Docker, 2019] poussant un peu plus loin la notion de mutualisation : les conteneurs légers (par opposition à la VM, parfois qualifiée de conteneur lourd). Cette approche vise à mutualiser le système d'exploitation et certaines librairies allégeant ainsi considérablement la granularité des briques déployées ce qui permet de gagner en rapidité de déploiement (cf. Figure 2.5). Un conteneur léger peut être déployé aussi bien sur des PMs que sur des VMs. L'approche conteneurs légers reste parfois critiquée en termes de sécurité et d'isolation par rapport aux VMs. En effet, il n'est pas possible à l'heure actuelle d'isoler les ressources (i.e. CPU, RAM, etc.) utilisées par des conteneurs basés sur la même machine, ce qui peut amener un conteneur à monopoliser les ressources de ses voisins et ainsi déstabiliser l'ensemble du système. La dernière version de *Docker* introduit la pos-

sibilité d'appliquer une limite maximum de consommation de ressource pour l'ensemble des conteneurs d'une même machine. Bien que cela évite les dérives, il n'est pas encore possible d'allouer de manière fine et dynamique des ressources aux conteneurs en fonction de leur besoin via l'API de haut niveau. Ceci est toutefois contournable en s'appuyant sur la fonctionnalité *cgroups* (i.e. *control groups*) du noyau Linux dans le but de limiter, compter ou isoler l'utilisation des ressources ou encore de prioriser celles-ci pour certains groupes.

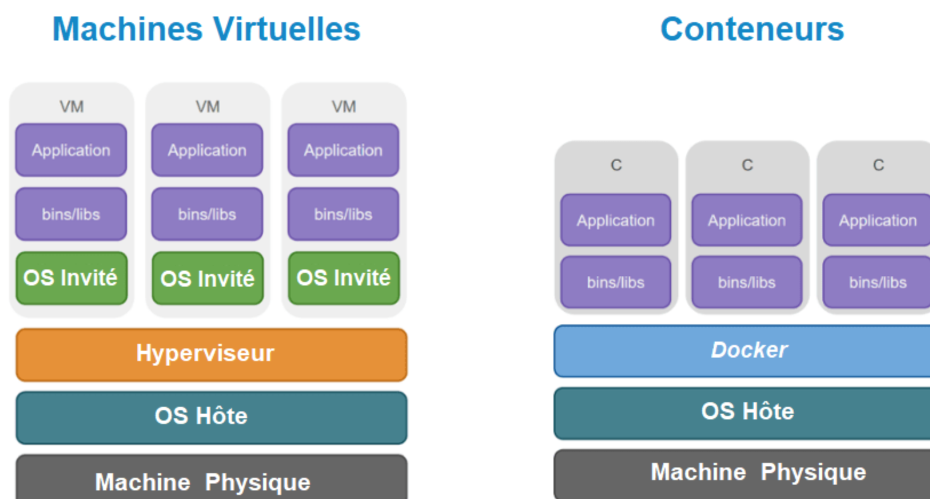


FIGURE 2.5 – Architectures des machines virtuelles et des conteneurs Docker

2.2.7 Modèle économique

Grâce à la mutualisation des ressources informatiques, le Cloud permet des économies d'échelle pour les fournisseurs de service. En effet, la mutualisation de la demande associée aux atouts de la virtualisation, notamment en termes de flexibilité, favorisent une exploitation maximale des ressources disponibles au niveau des PMs. D'autres avantages doivent être pris en compte tels que la possibilité pour les gestionnaires de centres de données de bénéficier de remises quantitatives sur le matériel utilisé ou encore la possibilité d'automatiser un certain nombre de tâches d'administration. Cela permet, en sortie, de réduire la somme totale des coûts associés. D'un autre côté, les usagers des services du Cloud bénéficient également d'avantages financiers. En effet, la souplesse du Cloud assure un accès facile et quasi immédiat à des ressources informatiques sans investissement important (e.g. acquisition de matériel, de licences, etc.) et sans engagement à long terme. De plus, les coûts d'exploitation associés à l'utilisation de ces services sont amoindris (e.g. frais de maintenance, de mise à jour, etc.). De cette manière, les usagers prennent moins de risques en s'appuyant sur un fournisseur plus expérimenté pour la réalisation de certaines fonctionnalités. Cette option peut finalement s'avérer plus rentable du fait qu'ils bénéficient eux aussi des économies d'échelles réalisées par les fournisseurs.

Choix d'offres. Les fournisseurs IaaS proposent une large gamme d'instances (VMs) en fonction des besoins des utilisateurs. Concrètement, il s'agit de multiples combinaisons en matière de capacités CPU, RAM, stockage et réseau. Chaque type d'instance propose une ou plusieurs tailles possibles en fonction des exigences liées à au type de la tâche ciblée. Cela varie d'une micro instance avec 1 vCPU et 1 Gio de mémoire à une 10xlarge instance

avec 40 vCPU et 160 Gio de mémoire [Amazon, 2019]. la Figure 2.6 donne un aperçu de la tarification horaire d'*Amazon EC2* en fonction des offres de VM. La facturation proposée par Microsoft Azure est illustreé dans la Figure 2.7 [Microsoft, 2019].

	vCPU	ECU	Mémoire (Go)	Stockage des instances (Go)	Utilisation de Linux/UNIX
Usage général – Génération actuelle					
a1.medium	1	s.o.	2 Gio	EBS uniquement	0,0255 USD par heure
a1.large	2	s.o.	4 Gio	EBS uniquement	0,051 USD par heure
a1.xlarge	4	s.o.	8 Gio	EBS uniquement	0,102 USD par heure
a1.2xlarge	8	s.o.	16 Gio	EBS uniquement	0,204 USD par heure
a1.4xlarge	16	s.o.	32 Gio	EBS uniquement	0,408 USD par heure
t3.nano	2	Variable	0,5 Gio	EBS uniquement	0,0052 USD par heure
t3.micro	2	Variable	1 Gio	EBS uniquement	0,0104 USD par heure
t3.small	2	Variable	2 Gio	EBS uniquement	0,0208 USD par heure
t3.medium	2	Variable	4 Gio	EBS uniquement	0,0416 USD par heure
t3.large	2	Variable	8 Gio	EBS uniquement	0,0832 USD par heure
t3.xlarge	4	Variable	16 Gio	EBS uniquement	0,1664 USD par heure
t3.2xlarge	8	Variable	32 Gio	EBS uniquement	0,3328 USD par heure

FIGURE 2.6 – Illustration de la tarification horaire par type d'instance de VM par *Amazon*

INSTANCE	vCPU	RAM	STOCKAGE TEMPORAIRE	À L'UTILISATION
A1 v2	1	2,00 Gio	10 Gio	0,031 €/heure
A2 v2	2	4,00 Gio	20 Gio	0,065 €/heure
A2m v2	2	16,00 Gio	20 Gio	0,084 €/heure
A4 v2	4	8,00 Gio	40 Gio	0,135 €/heure
A4m v2	4	32,00 Gio	40 Gio	0,176 €/heure
A8 v2	8	16,00 Gio	80 Gio	0,281 €/heure
A8m v2	8	64,00 Gio	80 Gio	0,369 €/heure

FIGURE 2.7 – Illustration de la tarification horaire par type d'instance de VM par MS *Azure*

Paiement à l'usage. Le modèle de facturation à l'usage, communément appelé *pay-per-use* ou selon le modèle *pay-as-you-go*, est l'une des caractéristiques du Cloud. Ce modèle de facturation assure le client de payer exclusivement ce qu'il consomme réellement au moment où il consomme. On distingue de nombreuses offres Cloud de complexité variable. Au niveau du modèle de service IaaS, la tarification est un sujet assez complexe à cause de la difficulté de comparaison des offres. Pour la plupart des IaaS publics du marché, la tarification correspond à une heure d'instance consommée pour chaque instance. Cependant, en sachant que chaque heure partielle consommée est entièrement facturée (comme

proposé par *Amazon* [Amazon, 2019]), l'utilisateur peut être amené à payer davantage que ce qu'il consomme réellement s'il ne libère pas les ressources à temps. De ce fait, de plus en plus de fournisseurs d'IaaS proposent des modèles de tarification à granularité plus fine, comme dans le cas de Windows Azure et sa facturation en minutes pleines [Microsoft, 2019].

2.2.8 Efficience et Cloud Computing

L'efficience désigne le fait d'assurer un maximum de productivité pour un minimum d'efforts ou de coûts. La définition générale de l'efficience d'un système est donnée par :

≪ *Efficiency (of a system or machine) : achieving maximum productivity with minimum wasted effort or expense.* ≫ (Oxford Dictionary).

En informatique, on distingue généralement trois objectifs principaux couverts par la notion d'efficience : *resource efficiency*, *energy efficiency* et *cost efficiency* qui visent à maximiser la productivité tout en minimisant respectivement la quantité de ressources utilisée, la consommation énergétique et les coûts. Ces trois termes sont corrélés du fait qu'une optimisation de l'utilisation des ressources entraine une réduction de l'empreinte énergétique, elle-même synonyme de diminution des coûts. Dans ce contexte, le cloud apporte une utilisation plus efficiente des serveurs en les concentrant dans des *datacenters*, notamment par le biais de la mutualisation des ressources ainsi que la technologie de virtualisation. En effet, le modèle basé nuage se veut plus *éco-responsable* (càd. Efficient énergétiquement), notamment en termes de consommation d'énergie, que les architectures de modèle *Cluster* où les serveurs sont dédiés à des utilisateurs précis sans partage de ressources. la Figure 2.8 [Google, 2019] illustre l'optimisation de l'utilisation des ressources au niveau de ces deux approches, notamment en termes de nombre des serveurs déployés et leur taux d'utilisation. Il n'en reste pas moins que l'évolution de la consommation énergétique des centres de données reste préoccupante [Koomey, 2011].

En réalité, mesurer et contrôler l'efficience du Cloud notamment sur le plan énergétique s'avèrent être des tâches compliquées du fait que l'on s'intéresse à un large panel de ressources déployées (matériel informatique, systèmes de refroidissement, composants logiciels, réseau, etc.) [Mastelic et al., 2015]. Un indicateur technique universellement reconnu est le PUE (*Power Usage Effectiveness*) [Carlson et al., 2012] visant à évaluer l'efficience énergétique des *datacenters*. Il s'agit d'évaluer la quantité d'énergie totale consommée par le centre de données par rapport à la quantité d'énergie nécessaire au fonctionnement des équipements informatiques. Plus le résultat est proche du chiffre 1, plus le centre de données est considéré comme *éco responsable*. En outre, des efforts considérables ont été faits ces dernières années pour améliorer l'efficience énergétique du Cloud, par exemple en rendant les salles serveurs moins consommatrices de ressources : machines moins gourmandes, systèmes de refroidissements améliorés, optimisation des installations électriques, etc. De plus, de nombreux travaux se sont intéressés à optimiser l'utilisation des ressources informatiques dans les centres de données physiques [Verma et al., 2008a, Verma et al., 2008b].

Dans le contexte du Cloud, la majorité des efforts visant à améliorer l'efficience énergétique se sont focalisés sur côté le matériel (càd. L'infrastructure). D'autres efforts plus récents, par exemple [Sabharwal et al., 2013], considèrent que des gains énergétiques sont possible du côté du logiciel. Ces travaux tendent à rendre les couches logicielles

du Cloud plus « conscientes » des contraintes environnementales et ainsi accroître leur efficience énergétique.

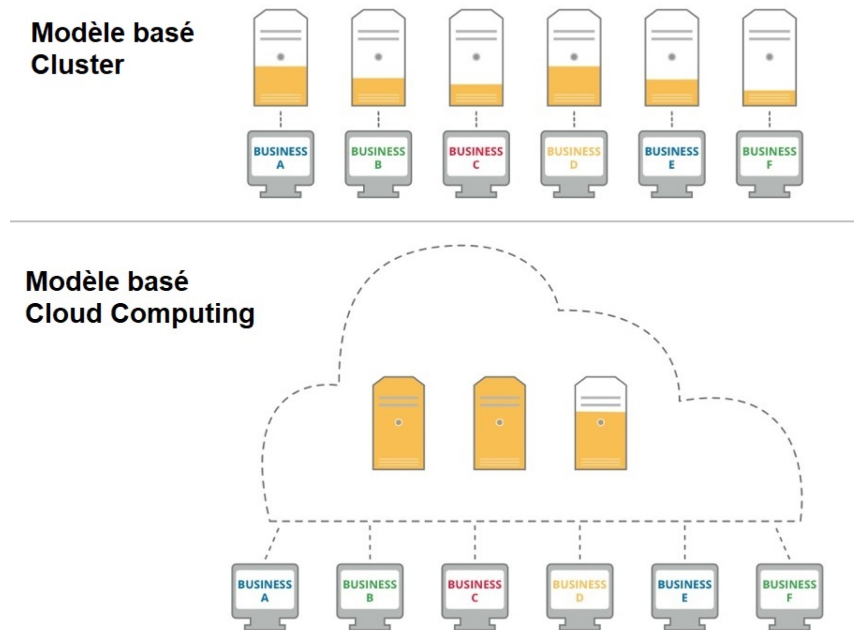


FIGURE 2.8 – Optimisation de l'utilisation des ressources : comparaison entre les modèles Cluster et Cloud.

2.3 Élasticité dans le cloud

Le Cloud Computing incarne un environnement hautement dynamique et variable. Dans ce contexte, l'élasticité, qui est l'une des caractéristiques fondamentales du Cloud, fournit des mécanismes qui permettent aux systèmes de supporter, au mieux, une montée de la charge de travail puis un retour à la normale, sans interruption de service et si possible sans répercussions sur la qualité du service. D'un autre côté, l'élasticité permet de respecter les engagements « *pay-per-use* » en fournissant des mécanismes de mise à l'échelle en cas de baisse de la charge, dans le but d'éviter les surfacturations inutiles. En d'autres termes, l'élasticité incarne l'un des rouages nécessaires à l'optimisation des ressources (i.e., optimisation de coûts), tout en assurant aux utilisateurs un niveau de service optimal (i.e., optimisation des performances), dans un contexte de plus en plus dynamique et variable.

Ainsi, l'élasticité permet aux fournisseurs de services cloud de répondre aux demandes de leurs usagers avec une réactivité maximale tout en tenant compte des multiples évènements imprévisibles dont sont sujets leurs applications.

2.3.1 Définitions

Au cœur du modèle Cloud, la notion d'élasticité introduit des concepts importants se rapportant à la notion de « passage à l'échelle » ou de « scalabilité ». En effet, on parle souvent de « capacité de montée en charge » pour définir la capacité d'un système à s'adapter aux dimensions de la tâche qu'il a à traiter. Dans le contexte du Cloud, la scalabilité correspond à la capacité d'une application à absorber une montée en charge

en ajoutant des ressources (CPU, RAM, stockage, réseau) afin de maintenir un certain niveau de performance et de qualité de service en fonction des besoins. Cependant, le terme « scalabilité » ne renvoie pas au caractère temporaire de cette augmentation de capacité [Weber et al., 2014]. Au final, l'élasticité implique la scalabilité, mais un système dit « scalable » n'est pas systématiquement « élastique ».

Selon le dictionnaire Larousse, la définition de l'élasticité est donnée par :

« *L'aptitude d'un corps à reprendre, après sollicitations, la forme et les dimensions qu'il avait avant d'être soumis à ces sollicitations.* »

Dans le contexte du Cloud Computing, de nombreuses définitions ont été proposées aussi bien par la communauté scientifique qu'industrielle. Nous retenons deux définitions récentes en gardant à l'esprit le préfixe « rapide » de cette propriété (*rapid elasticity*).

Définition 1 . Selon le NIST [Mell et al., 2011], l'élasticité rapide du cloud est définie par :
« *Rapid elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.* »

Définition 2 . Les auteurs de [Herbst et al., 2013], proposent une autre définition de l'élasticité :
« *Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible.* »

Outre la dimension temporelle inhérente à l'élasticité, ces deux définitions mettent l'accent sur certaines de ses propriétés. Il s'agit d'ajuster la quantité de ressources en fonction de la demande de manière réactive, précise, efficace et automatisée. Le tout, assuré avec une souplesse telle qu'à tout moment, l'utilisateur a l'illusion d'avoir une quantité infinie de ressources à sa disposition. Il est aussi question de dimensionnement (*scaling*) c'est-à-dire la mise à l'échelle en termes de ressources allouées, aussi bien à la montée qu'à la baisse de la demande. En d'autres termes, un système élastique, à partir d'une dimension donnée, a la capacité de « s'étirer » en ajoutant des ressources et de « se contracter » en en libérant afin de répondre à une demande, ici catégorisée par sa charge de travail en entrée.

Selon la classification proposée par [Galante and Bona, 2012], l'élasticité est caractérisée par le quadruplet : {niveau, stratégie, objectif, méthode} (voir Figure 2.9). Le niveau indique la cible de l'élasticité (infrastructure, plateforme, application). La stratégie décrit la politique (*policy*) de mise en œuvre de l'élasticité (proactive, réactive, hybride). L'objectif incarne le but visé par un redimensionnement (performance, coût, énergie). Enfin, la méthode spécifie le mécanisme d'élasticité exécuté (mise à l'échelle verticale/-horizontale, migration). Nous présentons dans ce qui suit les détails de ce quadruplet.

2.3.2 Niveaux d'action de l'élasticité

Les actions de mise à l'échelle de l'élasticité peuvent opérer à différentes portées ou niveaux. Une portée est définie selon la nature des ressources visées ou selon la couche (IaaS, Paas, SaaS) visée au sein du système Cloud :

— *Niveau infrastructure* : L'élasticité concerne principalement le niveau IaaS où les ressources approvisionnées sont généralement des instances de VMs. Néanmoins, d'autres ressources au niveau de l'infrastructure peuvent être concernées comme la capacité de stockage ou la bande passante.

— *Niveau plateforme* : L'élasticité consiste à ce niveau en des actions de dimensionnement liées aux conteneurs légers ou encore les bases de données.

— *Niveau application* : Les actions d'élasticité peuvent aussi bien concerner le niveau applicatif, notamment aux instances de services ou services web.

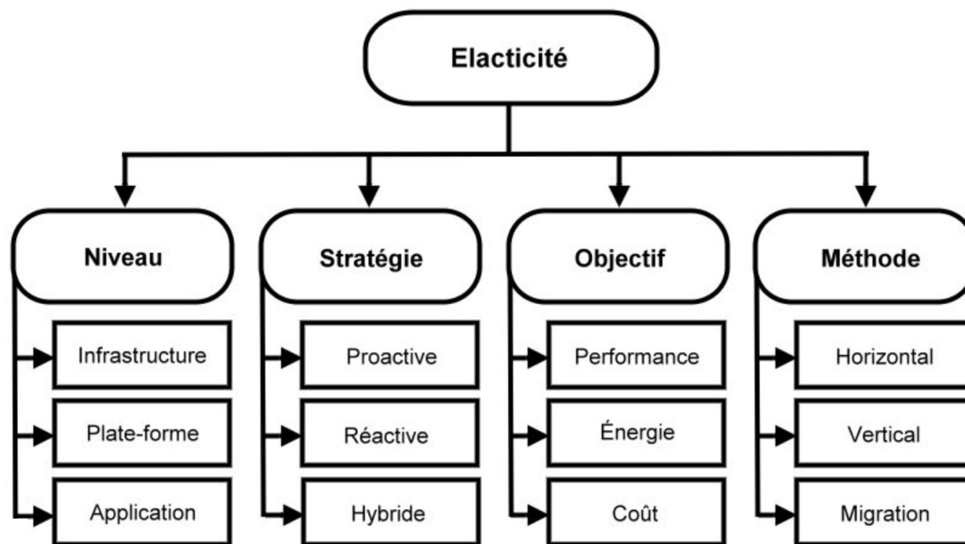


FIGURE 2.9 – Quadruplet de l'élasticité [Galante and Bona, 2012]

2.3.3 Stratégies d'élasticité et techniques sous-jacentes

Idéalement, le processus de dimensionnement inhérent à l'élasticité doit être automatisé afin d'assurer une gestion des ressources la plus efficace possible. On distingue trois stratégies de mise en œuvre du dimensionnement automatique (*autoscaling*) :

Stratégie de dimensionnement Réactif. Les stratégies d'élasticité pour dimensionnement réactif se basent sur la demande et plus particulièrement sur l'état courant du système. Une stratégie dite réactive réagit à certains évènements (e.g., changements d'états) mais ne les prévoit pas. Ce comportement s'appuie généralement sur un service de surveillance (*monitoring*) [Aceto et al., 2013] chargé de mesurer l'utilisation des ressources, et plus généralement l'état du système par rapport à sa charge de travail (*workload*). Ce service peut indiquer des situations nécessitant d'augmenter ou de réduire la quantité de ressources déployées dans le système. Le dimensionnement réactif se base généralement sur plusieurs techniques :

— *Les règles à base de seuils (threshold-based rules)* : afin d'augmenter/diminuer la capacité du système en ressources. Ces règles prennent la forme [**Si** condition(s) **Alors** action(s)] [Lorido-Botrán et al., 2012] où la partie condition(s) correspond à des métriques de consommation de ressources (e.g. taux d'utilisation du CPU) que l'on va confronter à des seuils supérieur et inférieur (*upper/lower thresholds*). La partie action(s) correspond aux actions de dimensionnement des ressources (ajout/retrait). Au niveau des stratégies réactives, on parle souvent de la notion de « période de calme » ou de « refroidissement » (*cooldown period*). Il s'agit concrètement de désactiver temporairement une règle après son déclenchement pour assurer une certaine stabilité du système (e.g., éviter des redimensionnements opposés en ajoutant une ressource puis en la retirant juste après).

— *La théorie du contrôle (control theory)* : cette théorie vise à analyser les propriétés d'un système en vue de l'amener d'un état initial donné à un état final souhaité par le biais de commandes (contrôle) et en respectant certains critères [Mendieta et al., 2017]. Un système de contrôle peut être vu comme un processus itératif visant à mesurer, comparer, appliquer et corriger l'état d'un système contrôlé afin de stabiliser le système et l'optimiser selon certains critères ou contraintes.

Stratégie de dimensionnement Proactif (*Prédictif*). Une stratégie proactive vise à prévoir les besoins à venir en termes de ressources requises. Contrairement à l'approche réactive, on s'intéresse ici à l'état futur du système au lieu de son état actuel. Le but est d'anticiper la demande afin de fournir les ressources suffisantes à l'avance. Ce type de dimensionnement repose principalement sur plusieurs techniques scientifiques :

— *Théorie des files d'attente (queuing theory)* : cette théorie provenant du domaine des probabilités vise à étudier les solutions optimales de gestion des files d'attente et s'appuyant sur des modèles analytiques [Ali-Eldin et al., 2012]. Une file d'attente se formant lorsque l'offre d'un système (ressources) est inférieure à sa demande (clients). L'analyse des files d'attente vise à caractériser le degré de performance du système (temps de réponse, disponibilité, débit) afin de prédire son comportement. Cette technique est également utilisée pour les stratégies réactives [Yataghene et al., 2014].

— *Apprentissage par renforcement (reinforcement learning)* : cette technique vise à doter les systèmes de comportements d'apprentissage automatique. Il s'agit d'apprendre d'expériences passées ce qu'il convient de faire en différentes situations [Sutton and Barto, 1998]. Concrètement, le système acquière de manière automatisée des compétences dans sa prise de décision en fonction des échecs (récompense négative) ou des succès (récompense positive) constatés. Ainsi, le système suit ce processus de manière itérative et tend à affiner la qualité de ses actions futures en apprenant de ses actions passées.

— *Analyse des séries chronologies (time series analysis)* : une série chronologique ou temporelle est une suite de valeurs numériques caractérisant l'évolution d'un valeur donnée (e.g., consommation des ressources) au cours du temps. Ces suites peuvent être formalisées mathématiquement afin de les analyser sous un spectre statistique ou probabiliste. L'idée dernière cette technique est d'étudier le comportement passé des séries enregistrées afin d'éventuellement en déterminer des tendances (*patterns*), dans le but d'en prévoir le comportement futur [Montgomery et al., 1990].

Stratégie de dimensionnement Hybride. Les stratégies d'élasticité hybrides combinent les stratégies de dimensionnement réactif et prédictif. Cette approche considère l'état courant du système ainsi que son état futur, en se basant sur les différentes techniques listées ci-dessus. Il s'agit d'être à la fois réactif aux changements tout en prévoyant à l'avance les besoins du système en termes de ressources et d'en anticiper les performances [Ali-Eldin et al., 2012].

2.3.4 Objectifs de l'élasticité

L'élasticité vise à gérer d'une manière précise l'ajout et le retrait de ressources Cloud afin d'assurer leur disponibilité de manière suffisante et optimale en fonction de la demande. Cela revient à éviter les cas de *sur-dimensionnement* et de *sous-dimensionnement* qui impactent aussi bien les fournisseurs que les usages du service.

Sur-dimensionnement (*over-provisionning*). Lorsque la quantité de ressources (e.g. nombre de VMs) attribué au système est supérieure à ses besoins (charge de travail), on dit que le système est en sur-dimensionnement ou encore en sur-approvisionnement (*over-provisioning*). Cela revient également à dire que le système est sous-chargé (*under-loaded*) en termes de capacité maximale de charge de travail (*workload*) qu'il peut absorber compte tenu de sa demande actuelle.

Sous-dimensionnement (*under-provisionning*). Lorsque la capacité en ressources attribuée au système est trop faible comparé à sa demande, on dit que le système est sous-dimensionné ou en cas de sous-approvisionnement (*under-provisioning*). En termes de sa capacité à absorber sa charge de travail, on considère que le système est sur-chargé (*over-loaded*).

la Figure 2.10 illustre l'évolution du *workload* (courbe noire) dans le temps ainsi que la quantité de ressources allouées au système et ses répercussions sur le plan financier et énergétique. La courbe orange montre la capacité en ressources d'une infrastructure de type *Cluster* (càd. Non élastique). La courbe bleue représente la quantité de ressources d'une infrastructure élastique de type Cloud. Dans le cas du modèle *Cluster*, on observe que l'infrastructure est constamment sur-dimensionnée mais ne permet pas d'absorber les importants pics de charges soudains, menant dans certains cas à un point de rupture se traduisant par l'interruption complète du service. En revanche, dans l'infrastructure Cloud, on peut voir que l'élasticité permet de s'adapter à la charge de travail en ajoutant et en retirant des ressources. Néanmoins, on constate que l'approvisionnement en ressources est tantôt supérieur à la demande réelle (zones en vert) et tantôt inférieur à celle-ci (zones en rouge).

Le sur-dimensionnement reflète l'inefficacité avec laquelle un système est utilisé et renvoie au gaspillage énergétique et donc financier lié à cette utilisation [Dashti and Rahmani, 2016]. Le fournisseur de service a donc tout intérêt à éviter le sur-dimensionnement en vue de limiter les coûts de mise en service et augmenter son profit. Cela se traduisant également en une diminution des coûts d'utilisation des ressources du point de vue de l'usager du service. D'un autre côté, le manque de ressources impacte directement les performances et la qualité du service fourni. Cela se manifeste notamment par des temps de réponse important ou encore l'indisponibilité du service. Par conséquent, le sous-dimensionnement entraine une insatisfaction des usagers du service cloud et de ses utilisateurs finaux, pouvant aboutir à une perte de clients et une baisse de profit pour le

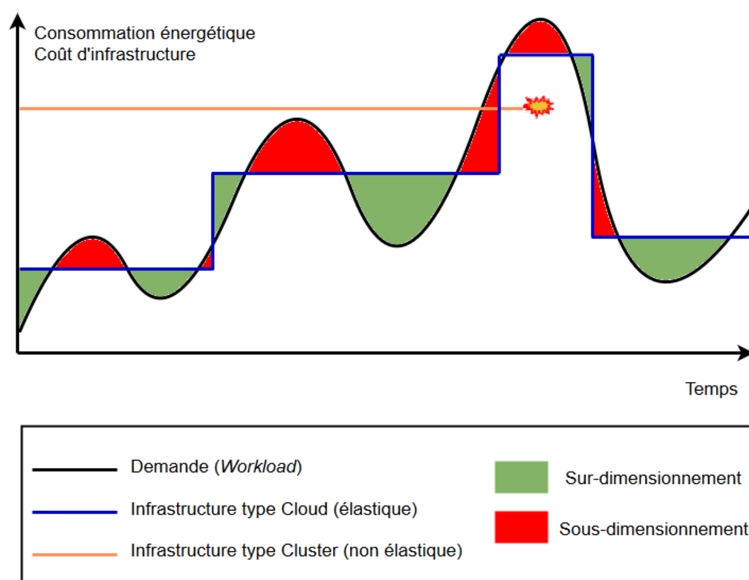


FIGURE 2.10 – Élasticité : sur/sous-dimensionnement dans les modèles Cluster et Cloud

fournisseur de service. Ce dernier a, dans ce cas, grand intérêt à éviter les cas de sous-dimensionnement prolongé en vue de préserver une certaine qualité de service et d'éviter les différentes pénalités associées à la violation de celle-ci.

2.3.5 Méthodes d'élasticité

L'élasticité introduit plusieurs mécanismes permettant à un système de s'adapter à sa charge de travail courante en ajoutant/retirant des ressources en vue d'éviter les cas de sur/sous-approvisionnement. Selon [Galante and Bona, 2012], on distingue trois types de dimensionnement (*scaling*) : le dimensionnement horizontal (*horizontal scaling*), le dimensionnement vertical (*vertical scaling*) et la migration.

Dimensionnement horizontal (*horizontal scaling*). Selon le niveau d'action visé, le dimensionnement horizontal ou l'élasticité horizontale (voir Figure 2.11) consiste à ajuster le nombre de ressources en fonction de la demande (e.g., ajout/retrait d'instances de VMs ou de services au niveau infrastructure ou application). Ce type de dimensionnement permet d'augmenter la capacité du système d'une manière théoriquement infinie [Andrikopoulos et al., 2013]. On retrouve également dans la littérature les termes *Scale-Out* et *Scale-In* pour indiquer respectivement l'ajout et le retrait de ressources en dimensionnement horizontal. En outre, l'élasticité horizontale est étroitement liée au mécanisme de répartition de charge (*load balancing*) qui consiste, comme son nom l'indique, en la répartition de la charge de travail entre les différentes entités déployées au sein du groupement (*pools*) de ressources du système. Concrètement, cela est accompli à travers un répartiteur de charge (*load balancer*) qui s'occupe de distribuer équitablement la demande entre les différentes ressources (VMs, instances de service).

Dimensionnement vertical (*vertical scaling*). Le dimensionnement vertical ou élasticité verticale concerne principalement le niveau infrastructure. Aussi appelé redimensionnement ou *resizing/redimensioning* en anglais, ce mécanisme consiste à augmenter/diminuer les ressources allouées à une VM existante telles que le CPU, la mémoire RAM ou la ca-

pacité de stockage [Yazdanov and Fetzer, 2012]. Concrètement, l'hyperviseur (cf. Section 2.2.6) redimensionne l'infrastructure virtualisée en (dés)allouant des ressources aux VMs. Cela implique que le dimensionnement vertical est limité par la quantité de ressources physiques disponibles sur la PM hôte. Les termes *Scale-Up* et *Scale-In* sont également utilisés dans la littérature pour indiquer respectivement l'ajout et le retrait de ressources (CPU, RAM, etc.) sur une VM existante.

la Figure 2.11, inspirée de [Dupont, 2016], illustre le dimensionnement horizontal et vertical d'un système au niveau infrastructure (en se concentrant sur les VMs).

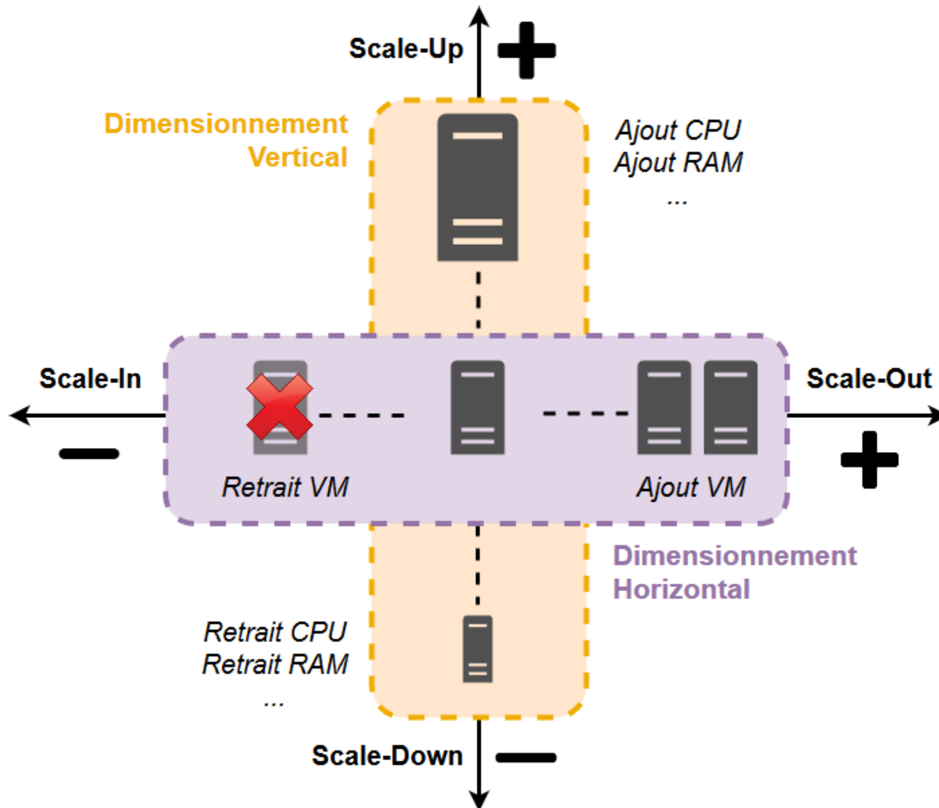


FIGURE 2.11 – Dimensionnement horizontal et vertical au niveau Infrastructure

Migration. La migration dans un système informatique consiste à déplacer une entité (selon le niveau d'action) d'un hôte (là où elle est déployée) à un autre dans le but d'optimiser la consommation des ressources dans le système et/ou la performance du service exécté. Dans le cas de l'élasticité au niveau infrastructure, il s'agit de déplacer une VM d'un hyperviseur à un autre afin de réorganiser la consommation de ressources dans le but d'éteindre des PMs superflues synonymes de gaspillage énergétique et financier [Voorsluys et al., 2009]. Dans le cas de l'élasticité au niveau application (ou SaaS), la migration consiste à déplacer une application d'une VM à une autre [Carrasco et al., 2017] afin de profiter d'une meilleure configuration en termes de ressources (e.g., une meilleure bande passante) ou encore pour bénéficier de remises avantageuses en termes de prix des VMs déployées, notamment en ciblant une VM hébergée chez un autre fournisseur de service proposant de meilleures offres [Wang et al., 2012].

Dimensionnement hybride. En plus du dimensionnement horizontal et vertical et de la migration, on peut retrouver dans la littérature un autre type de dimensionnement dit hybride. Ce dimensionnement fusionne les approches d'élasticité verticale et horizontale afin de fournir des solutions plus complexes pour la gestion des ressources et l'optimisation des couts. Précisément, selon le contexte du système et les besoins de l'usager, le dimensionnement hybride est très souvent présenté comme un moyen de fournir un compromis (*trade-off*) visant à maximiser les performances d'un système notamment en termes de disponibilité et fiabilité tout réduisant au mieux les coûts associés à la mise à l'échelle ainsi qu'à la surveillance (*monitoring*) du système. Les auteurs de [Suleiman et al., 2012] illustrent très bien ce concept à travers un cas d'utilisation (l'application MyShop) en comparant les trois approches de dimensionnement (horizontal, vertical et hybride) en termes de coûts, de fiabilité et de disponibilité.

2.4 Informatique autonomique

Devant la complexité croissante du Cloud et son environnement hautement dynamique (multiples facteurs déterminant, demande variable, consommation de ressources à ajuster), il devient quasiment impossible aux opérateurs humains de s'acquitter de tâches d'administrations permettant de réagir à ce contexte d'exécution afin de garantir la performance et la fiabilité du Cloud. Face à cette problématique, il devient donc primordial de doter les systèmes Cloud de capacités de d'auto-gestion de manière autonomique.

2.4.1 Définitions

L'informatique autonomique a été introduite par IBM [Horn, 2001] en 2001 en réponse à la difficulté d'administration des systèmes informatiques et leur complexité grandissante (en termes d'envergure, de technologies, d'architecture, etc.) ainsi que le contexte hautement dynamique dans lequel ils évoluent (sollicitations variables, panne logicielles/-matérielles, etc.). L'idée derrière l'informatique autonome est de soulager les opérateurs humains en les assistant dans les tâches d'administrations de plus en plus complexes. La philosophie de l'informatique autonomique s'inspire du fonctionnement du système nerveux humain. Ainsi, un système informatique autonomique, en intégrant certaines propriétés autonomiques, pourrait se comporter comme un corps humain (e.g. en régulant le rythme cardiaque en fonction de l'effort fourni). En 2003, *Kephart et al.* donnèrent une définition au domaine dans l'article *The vision for autonomic computing* [Kephart and Chess, 2003] : « *Autonomic computing is the ability of an IT infrastructure to adapt to change in accordance with business policies and objectives.* »

2.4.2 Propriétés autonomiques

Les systèmes autonomiques sont des systèmes auto-gérés, caractérisés par sept propriétés essentielles. Dans la littérature, on parle souvent de « propriétés auto-x » ou *self-x properties*. On en distingue deux types : les propriétés dites principales (auto-configuration, auto-guérison, auto-optimisation et auto-protection) et les propriétés considérées secondaires (auto-connaissance, sensibilité au contexte et auto-adaptation) [Horn, 2001, Kephart and Chess, 2003, Huebscher and McCann, 2008, Berns and Ghosh, 2009, Weiss et al., 2011] :

Auto-configuration (*self-configuration*). Cette propriété définit la capacité du système à s'adapter de manière dynamique aux changements de son environnement d'exécution. Cela est accompli en suivant des objectifs ou politiques (*policies*) de haut niveau qui décrivent les états désirables ou visés du système. Un système dit « auto-configurable » est à même de s'installer, de se configurer et de se mettre à jour de manière autonomique, en fonction de son état et de ses besoins, en respectant certaines recommandations prédéfinies par un opérateur humain.

Auto-guérison (*self-healing*). Un système doté de cette propriété est en mesure de détecter, diagnostiquer puis réparer/compenser des anomalies (e.g. pannes logicielles/matérielles) survenant au cours de son fonctionnement.

Auto-optimisation (*self-optimization*). Pourvu de cette propriété, un système est à même d'optimiser en permanence l'utilisation de ses ressources. L'objectif d'un tel comportement est de s'assurer que le système maintienne un état défini comme « optimal » par un opérateur humain. Ceci est notamment exprimé en termes de politiques de gestion prédéfinies, en spécifiant les critères concernés par l'optimisation (e.g. consommation énergétique, performance, coût, etc.).

Auto-protection (*self-protecting*). Un système se protège lui-même s'il est en mesure d'anticiper les problèmes ou évènements pouvant altérer son état et donc sa stabilité (e.g. pannes, etc.). Concrètement, le système ajuste son comportement afin de gérer tous types de menaces en y répondant directement ou en prenant les précautions adéquates. Cette propriété est étroitement liée à la sécurité.

Auto-connaissance (*self-knowledge*). Pour être en mesure de s'auto-gérer, un système doit à tout moment connaître ses composants, leurs états, leurs capacités et leurs liens avec d'autres systèmes. En outre, il doit prendre connaissance des ressources qui lui appartiennent, celles qui lui sont externes et enfin celles qui peuvent être partagées. On retrouve cette propriété également sous le nom de « auto-conscience » ou *self-awareness* dans la littérature.

Sensibilité au contexte (*context-awareness*). Un système autonomique est considéré sensible ou encore conscient de son contexte s'il est en mesure de percevoir et d'utiliser les différentes informations relatives à son environnement. Précisément, il s'agit de prendre connaissance de la localisation, du temps, de la température ou de l'identité des utilisateurs, ressources et systèmes présents dans son contexte afin d'adapter dynamiquement ses fonctionnalités selon les besoins prédéfinis et inhérents à ce contexte.

Auto-adaptation (*self-adapting*). L'auto-adaptation est une notion très large fortement liée aux propriétés précédemment introduites. Un système peut s'adapter de différentes manières selon son état, l'état de son environnement et devant les différents évènements pouvant se produire durant son fonctionnement. La capacité d'auto-adaptation est souvent liée aux principes de « décision » et de « contrôle » [Weiss et al., 2011] où le système prend en compte une multitude de paramètre afin d'appliquer les actions les plus appropriées selon les conditions. Par exemple, devant une montée brutale de la demande, un système auto-adaptatif doit réagir de manière à allouer plus de ressources afin de maintenir un certain niveau de service. Cet exemple simple, du moins en apparence, implique

la satisfaction de plusieurs propriétés auto-x. En l'occurrence, l'allocation de ressources revient à optimiser la consommation de celles-ci (*self-optimization*). En même temps, on constate que le système est en mesure de détecter non seulement une montée de la charge, mais aussi l'impact qu'elle peut avoir sur ses ressources (*context-awareness* et *self-knowledge*). Aussi, le fait d'allouer plus de ressources équivaut à une reconfiguration (*self-configuration*). En outre, assurer un certain niveau de service peut être considéré comme une mesure prise, dans le but d'éviter l'arrêt du service et d'assurer son bon fonctionnement (*self-protecting*). Enfin, si on considère la montée de la charge comme une anomalie impactant la stabilité du système (i.e. baisse des performances), le fait d'y remédier en allouant plus de ressources équivaut à de l'auto-guérison (*self-healing*).

2.4.3 Boucle de contrôle autonomique

Afin de mener à bien les différentes tâches d'auto-administration caractérisant leur comportement, les systèmes autonomiques se basent généralement sur des systèmes de gestion autonomiques [Huebscher and McCann, 2008]. Le modèle le plus populaire utilisé à ces fins, est la boucle de contrôle fermée (*closed control loop*) connue sous le nom de MAPE-K pour (*Monitor, Analyze, Plan, Execute – Knowledge*). Ce modèle (voir Figure 2.12) a été introduit par IBM en 2005 [Group and others, 2005] en s'inspirant des boucles de contrôle utilisée dans le domaine de l'automatique. Ainsi, l'élément géré (système, application) n'est plus administré par un opérateur humain mais par un processus adaptatif constitué d'une succession d'opérations de surveillance (monitoring), de calculs (analyse et planification) et de reconfigurations (actions), devant être réalisées de façon continuelle afin suivre les besoins de l'application et d'en assurer une configuration constamment optimale [Letondeur, 2014].

L'élément géré (*managed element*). L'élément géré représente une entité qu'on souhaite doter de comportements d'auto-gestion en lui associant un gestionnaire autonomique. Concrètement, il s'agit d'un système ou d'une partie d'un système (logicielle ou matérielle) que l'on souhaite administrer de manière autonomique par le biais de la boucle MAPE-K. La communication entre l'élément géré et le gestionnaire autonomique est assurée au travers de deux interfaces constituées respectivement de capteurs et d'actionneurs :

- *Les capteurs (sensors)* : également appelés sondes (ou *probes* en anglais) dans la littérature, les capteurs assurent la collecte d'informations et métriques sur l'élément géré. Ces informations sont généralement indicatives de l'état du système (e.g. ressources allouées, charge de travail, violations enregistrées, etc.) et sont exposées au gestionnaire autonomique.
- *Les actionneurs (actuators)* : également connus sous l'expression *effectors* dans la littérature, les actionneurs représentent des points d'entrée à l'élément géré sous différentes granularités. Il s'agit concrètement de leviers possibles pour reconfigurer le système (e.g. ports d'écoute, accesseurs, etc.). L'ensemble des actionneurs constitue l'API d'accès et de reconfiguration de l'entité gérée.

Le gestionnaire autonomique (*autonomic manager*). Le gestionnaire autonomique est alimenté par les métriques collectées par les multiples capteurs de l'élément géré, lui offrant les informations nécessaires sur l'état global du système. Il analyse toutes les informations à sa disposition et décide d'appliquer une ou plusieurs actions d'auto-administration (selon les objectifs préalablement spécifiés) sur l'élément géré (e.g. opti-

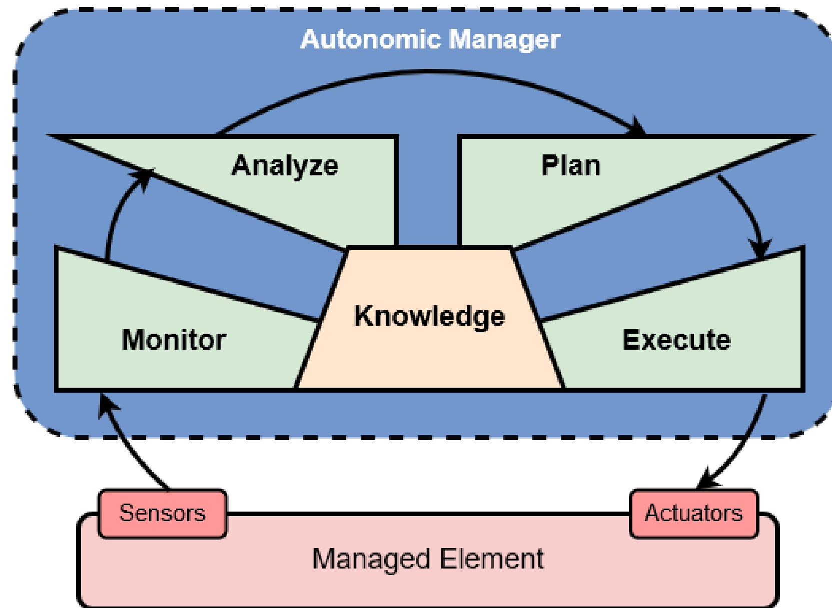


FIGURE 2.12 – Boucle de contrôle autonomique MAPE-K

misation, (re)configuration, guérison, protection, etc.). Enfin le gestionnaire autonomique applique le résultat de son diagnostic via les actionneurs. Nous détaillons ci-dessous les différentes phases et composantes caractérisant le fonctionnement de la boucle MAPE-K :

— *Phase d'observation (Monitor - M)* : La première phase concerne l'observation du système géré. Dans un premier temps, elle se charge de capturer les différentes informations et métriques des ressources concernées par le biais des différents capteurs déployés. Ensuite, elle transmet les données recueillies au gestionnaire autonomique qui prendra alors conscience de l'état global du système.

— *Phase d'analyse (Analyze - A)* : La deuxième phase de la boucle est en charge de considérer les différentes données enregistrées lors de la phase d'observation pour les confronter aux politiques et stratégies globales de gestion préalablement définies. L'objectif de cette phase est d'identifier les actions à entreprendre et les changements à apporter au système en vue d'améliorer et d'affiner son état, de manière à atteindre un état optimal attendu.

— *Phase de planification (Plan - P)* : à partir du diagnostic rendu par la phase d'analyse, la planification se charge de produire un plan complet et précis des actions à mettre en œuvre sur le système. Le plan fourni considère généralement les contraintes temporelles, les effets de bords et les répercussion négatives/positives qu'entraine une action sur l'ensemble du système. Les phases d'analyse et de planifications sont parfois regroupées en une unique phase dite de décision.

— *Phase d'exécution (Execute - E)* : Le plan d'action produit par la phase de planification peut concerner plusieurs entités composant le système. La phase d'exécution se charge d'appliquer les différentes actions du plan. Cela consiste à appeler les différents actionneurs qui opèrent les modifications nécessaires sur le(s) élément(s) géré(s). La phase d'exécution incarne le résultat du traitement effectué par la boucle MAPE-K à l'issu d'une itération de celle-ci.

— *Base de connaissances (Knowledge - K)* : Les différentes phases de la boucle MAPE-K sont reliées à une base de connaissances. Celle-ci peut contenir un large panel

d'informations relatives au système et son environnement. Pouvant être alimentées par différentes sources, la base de connaissance regroupe des informations telles que les historiques de la phase d'observation, les différentes décisions de reconfigurations ou encore des informations rajoutées par des opérateurs humains dans le but d'assister et de guider le gestionnaire autonomique dans sa prise de décision.

2.4.4 Contrôleur d'élasticité autonomique dans les systèmes Cloud

L'élasticité caractérise la capacité d'un système informatique à s'adapter aux variations de sa charge de travail en entrée, en provisionnant ou en libérant des ressources informatiques de manière autonomique, d'une façon telle qu'à tout moment, les ressources disponibles correspondent le plus possible à la demande rencontrée. Dans le contexte du Cloud Computing (voir Figure 2.13), les comportements élastiques autonomiques sont généralement pris en charge par un gestionnaire autonomique appelé contrôleur d'élasticité (*elasticity controller*).

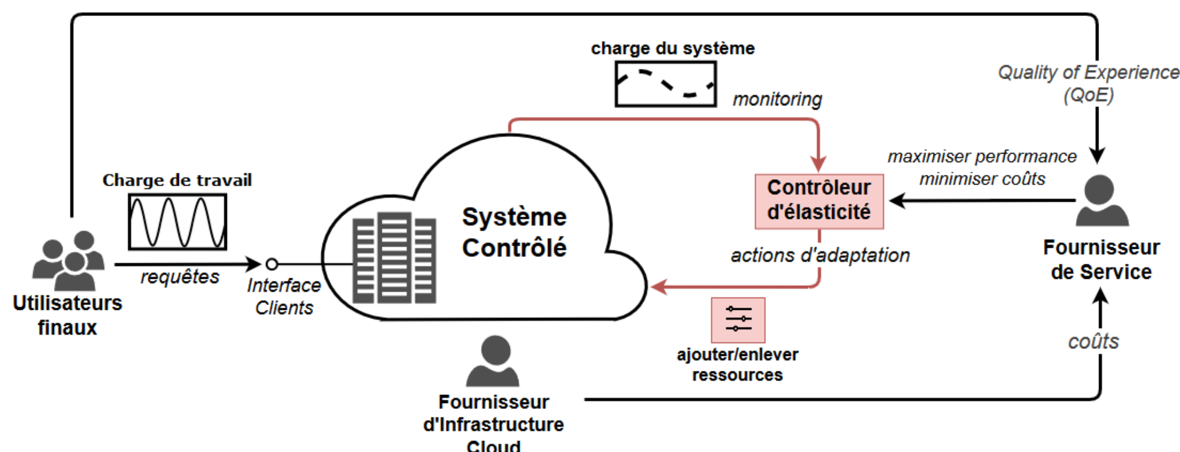


FIGURE 2.13 – Vue d'ensemble du comportement élastique autonomique d'un système Cloud

L'élément géré est le système Cloud dans sa globalité et sa surveillance (monitoring) est assurée à plusieurs niveaux et à différentes échelles. Cette surveillance revient à prendre conscience de l'état des différentes entités composant le système, ce qui permet d'en déterminer l'état global en termes d'élasticité. L'état global décrit la charge du système (*system load*) et en indique le sous/sur-dimension-nement (sur/sous-charge – *over/under-loading*) en termes de ressources allouées vis-à-vis de la charge de travail en entrée (*workload*).

Le fournisseur de l'infrastructure cloud communique des coûts d'utilisation au fournisseur du service cloud, qui y déploie ses applications, en fonction des ressources dédiées à son service. Le fournisseur de service reçoit également des retours de la part des utilisateurs finaux de son produit, décrivant la qualité de l'expérience (notamment en termes de performances) rencontrée lors de l'utilisation du service (*Quality of Experience* - QoE). Dans ce cas de figure, le fournisseur du service Cloud définit des politiques de haut niveau (*maximiser performance, minimiser couts,* etc.) afin d'administrer le dimensionnement des ressources selon ses contraintes financières et sa volonté d'assurer une bonne qualité de service à ses clients finaux. Le contrôleur d'élasticité, au centre du comportement autonomique du système cloud en termes de dimensionnement dynamique des ressources, intègre les politiques de haut niveau définies afin d'appliquer les actions d'adaptation (ajouter/retirer ressources) en fonction de l'état du système Cloud contrôlé.

2.5 Conclusion

Dans cette première partie de l'état de l'art, nous avons présenté le contexte dans lequel s'inscrit le sujet de la thèse en introduisant de nombreuses définitions, concepts de bases et technologies. Premièrement, nous avons présenté le modèle Cloud Computing et les grandes définitions qui lui sont liées. Ensuite, nous avons introduit, développé et détaillé la notion d'élasticité, se trouvant au cœur de nos travaux de thèse. Dans le contexte hautement complexe, dynamique et variable lié au fonctionnement du Cloud, nous avons exposé le domaine de l'informatique autonomique à travers ses caractéristiques et ses intentions, notamment dans le but de doter les systèmes informatiques de capacités d'auto-administration. Enfin, nous avons présenté le contrôleur autonomique d'élasticité au centre des comportements élastiques d'un système Cloud.

Chapitre 3

Travaux existants sur l'élasticité dans le Cloud

L'objectif de ce Chapitre est de recenser et d'analyser plusieurs travaux liés à l'élasticité dans le Cloud. Dans la Section 3.1, nous nous intéressons à quelques environnements conçus pour la gestion autonomique de l'élasticité. Dans la Section 3.2, nous nous penchons principalement sur quelques approches et modèles formels ayant proposé des solutions pour la spécification et la vérification de l'élasticité dans le Cloud.

Sommaire

3.1 Environnements pour la gestion autonomique de l'élasticité dans le Cloud . 34
3.1.1 Synthèse 39
3.2 Modèles formels pour l'élasticité dans le Cloud 40
3.2.1 Synthèse 44
3.3 Conclusion 46

3.1 Environnements pour la gestion autonomique de l'élasticité dans le Cloud

Au cours des dernières années, un effort considérable a été consacré à la gestion autonomique des ressources informatiques dans le Cloud en termes d'élasticité. Plusieurs contributions concernant la gestion et la planification de l'élasticité ont été proposées. Les solutions abordées dans cette Section fournissent des contrôleurs d'élasticité autonomiques basés sur des stratégies proactives et/ou réactives.

Les auteurs de [Letondeur, 2014] ont proposé un Framework pour la gestion autonomique de l'élasticité des applications dans le Cloud nommé Vulcan. En particulier, ils ont proposé une boucle de contrôle autonomique fermée basé sur le modèle MAPE-K (Monitor, Analyze, Plan and Execute) de IBM permettant d'assurer l'élasticité tout en considérant des scénarios complexes dans les systèmes Cloud. Cette boucle de contrôle consiste en un ensemble de composants. Premièrement, le composant Planification qui décrit comment une application doit se reconfigurer en fonction d'une décision d'élasticité. Comme le montre la Figure 3.1, lorsque le composant de planification reçoit une décision d'élasticité d'un composant Analyzer, il calcule le nouvel état de l'application à contrôler (nouvelle architecture de l'application). Pour accomplir cette tâche, il utilise une description initiale qui représente l'architecture actuelle de l'application et une autre description qui consiste en toutes les architectures possibles de l'application. Un algorithme

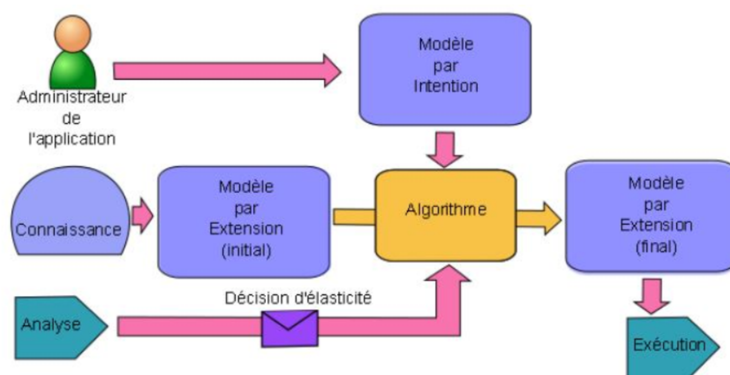


FIGURE 3.1 – Vulcan, un gestionnaire de planification de l'élasticité Vulcan [Letondeur, 2014]

associé utilise ces deux descriptions pour calculer et déterminer les modifications à effectuer dans l'architecture de l'application selon la décision d'élasticité. Ces modifications consistent principalement à l'ajout/suppression d'une machine virtuelle ou conteneur ainsi que d'autres opérations pour la reconfiguration de l'architecture actuelle.

Les auteurs de [Al-Dhuraibi, 2018] ont proposé un Framework appelé OCCI (MoDEMO) afin de relever le défi de la gestion de l'élasticité dans le Cloud pour minimiser les états de sur-dimensionnement et de sous-dimensionnement. Ce Framework prend en charge les méthodes d'élasticité verticale et horizontale, différentes techniques de virtualisation (VMs et conteneurs) et supporte plusieurs fournisseurs de Cloud. Les auteurs ont également proposé une approche appelée ElasticDocker pour la gestion de l'élasticité (verticale, migration) des conteneurs. Cette approche permet de combiner des contrôleurs d'élasticité qui opèrent au niveau des machines virtuelles et des conteneurs, comme montré dans la Figure 3.2. Les systèmes Cloud gérés sont surveillés de manière indépendante au niveau des VMs (niveau IaaS) et des conteneurs qui y sont déployés (PaaS). Les conteneurs sont surveillés via un composant VM Monitor et les conteneurs, à travers un composant Container Monitor. Les entités aux niveaux considérés (VMs, conteneurs) sont gérées par les composants VM VE Controller et Container VE Controller, qui contrôlent leur élasticité et enfin, les composants VM Execute et Container Execute s'occupent d'appliquer les actions d'élasticité décidées par les contrôleurs. Dans ces travaux, les contrôleurs d'élasticité VM VE et Contrainer VE communiquent afin d'affiner les prises de décision en termes d'élasticité, ce qui permet d'assurer une élasticité en multi-couches, au niveau infrastructure et plateforme.

Les auteurs de [Dupont, 2016] ont proposé un Framework autonomique de gestion de l'élasticité multi-couches (au niveaux infrastructure et application). Ils introduisent SCUBA, un gestionnaire autonomique de type MAPE-K permettant d'assurer le processus de gestion de l'élasticité multi-couches de bout en bout, depuis son paramétrage par l'administrateur Cloud jusqu'à son exécution. Ce Framework autonomique adopte une vision de type MAPE-K et s'appuie sur différents modèles : (1) un modèle conceptuel de l'élasticité multi-couches, permettant de représenter un système Cloud sous la forme d'un graphe de ressources de différents types (IaaS, SaaS). (2) Un modèle de surveillance, à travers l'outil perCEPtion, permettant d'identifier les points de contrôle impactant la décision de l'élasticité. (3) Un modèle d'adaptation, à travers le langage dédié ElaScript, qui repose sur le principe de tactiques d'élasticité, permettant de décrire les actions

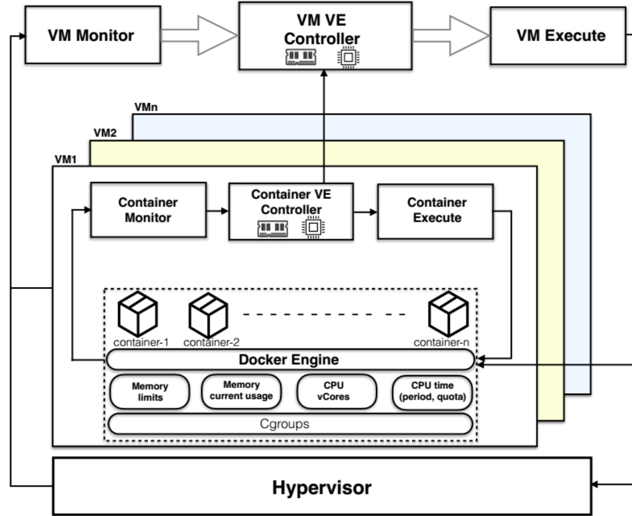


FIGURE 3.2 – Coordination de contrôleurs d'élasticité entre VMs et conteneurs Docker [Al-Dhuraibi, 2018]

à exécuter en fonction des symptômes constatés dans le système et (4) un modèle de décision d'adaptation, permettant de spécifier les préférences de l'utilisateur sous forme de stratégies impactant la prise de décision quant à l'élasticité. L'architecture du Framework est donnée dans la Figure 3.3.

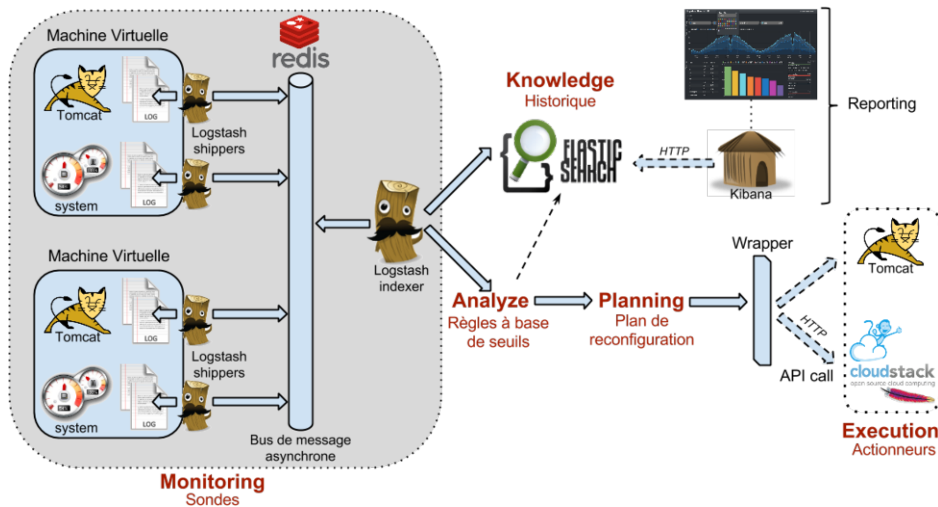


FIGURE 3.3 – Architecture générale du Framework SCUBA [Dupont, 2016]

Les auteurs de [Kranas et al., 2012] ont proposé ElaaS (Elasticity-as-a-Service), un Framework pour la gestion autonomique de l'élasticité du Cloud adressant les différents modèles de service (couches IaaS, PaaS et SaaS). Le Framework intègre cinq composants différents pour le traitement de l'élasticité, comme le montre la Figure 3.4. Ces composants sont unitairement accessibles au moyen de web services. Le composant Core de ElaaS coordonne les activités et les processus. Le composant Application Manager recueille et analyse les informations relatives à l'utilisateur et l'application. Le composant Monitoring Manager communique avec les sources de surveillance (sondes) qui peuvent être installées dans les différents niveaux Cloud (infrastructures, plate-forme ou application).

Le composant décideur dans le Framework est nommée Business Logic Manager. Enfin, le composant Action Manager envoie des messages d'action pour les composants appropriés d'une application et/ou plate-forme et produit un nouveau graphe de déploiement selon l'action exécutée. Par conséquent, l'élasticité est assurée selon le graphe de déploiement résultat.

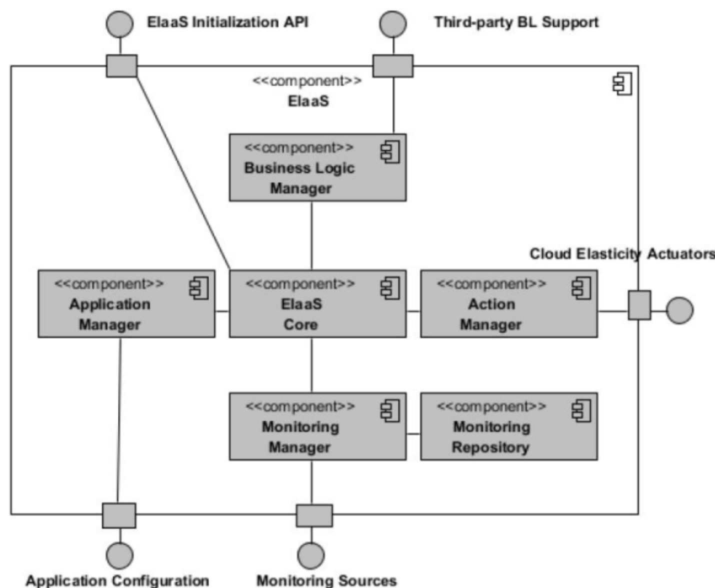


FIGURE 3.4 – Architecture du Framework ElaaS [Kranas et al., 2012]

Dans [Loff and Garcia, 2014] , les auteurs ont proposé Vadara, un Framework générique qui fournit une couche d'abstraction pour les fournisseurs de services Cloud existants. Ce Framework permet l'utilisation et le développement de stratégies d'élasticité de manière unifiée. Il permet de découpler les stratégies d'élasticité des plateformes Cloud sous-jacentes, rendant possible l'utilisation des stratégies génériques. Pour fournir des comportements élastiques, les auteurs proposent l'ajout d'une extension aux plateformes Cloud afin de permettre l'interaction entre les composants du fournisseur (par exemple, suivi et mise à l'échelle des composants) et les composants du Framework. Comme le montre la Figure 3.5, Vadara déploie un ensemble de composants : un composant Core qui configure et initialise les éléments du Framework. Un composant Monitor qui collecte, regroupe et soumet les informations de suivi à un composant Decider. Ce dernier analyse les informations de suivi et envoie les actions d'élasticité appropriées à un composant Scaler, qui fait appel au service de mise à l'échelle du PaaS afin d'exécuter les actions d'élasticité choisies par le Decider.

Les auteurs de [Moldovan et al., 2015] ont présenté MELA, un Framework pour la surveillance et l'analyse de l'élasticité des services basés Cloud. Ce Framework offre la possibilité aux développeurs et fournisseurs de services de surveiller et d'analyser le comportement des différentes ressources Cloud (VMs, applications, etc.) en vue de contrôler l'élasticité du système dans sa globalité. Les auteurs présentent MELA comme un service de surveillance et d'analyse de l'élasticité à la demande (elasticity space monitoring and analysis as a service). Ils proposent un ensemble de métriques à surveiller pour l'analyse d'un comportement élastique. Ils regroupent ces différentes métriques selon trois catégories (elasticity dimensions) : Cost, Quality et Resource (Coûts, Qualité, Ressources). La Figure 3.6 donne un aperçu de cette catégorisation. En se basant

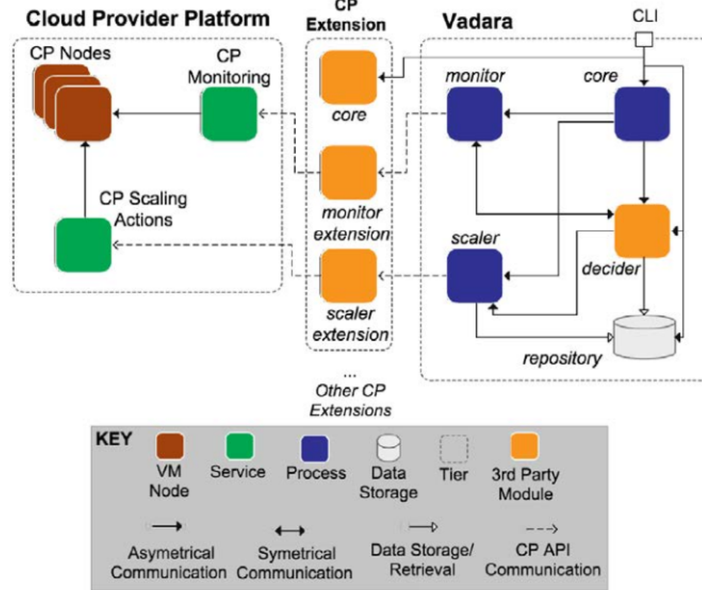


FIGURE 3.5 – Architecture du Framework Vadara [Loff and Garcia, 2014]

sur ce modèle, l'utilisateur de MELA (e.g. administrateur, développeur) peut définir les métriques concernées pour différents types de ressources. De plus, celui-ci peut spécifier ses besoins d'élasticité sous forme de seuils associés aux métriques lui permettant de capturer le comportement élastique de celles-ci. La solution offre aussi la possibilité de visualiser dynamiquement la topologie des ressources ainsi que l'état courant des différentes métriques observées.

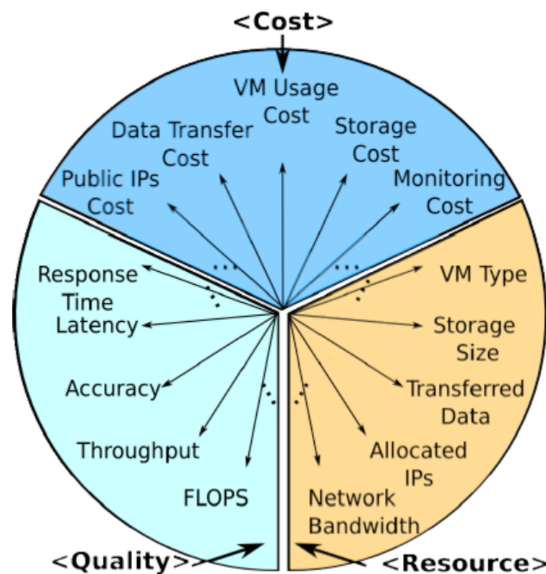


FIGURE 3.6 – Métriques de surveillance de l'élasticité de MELA [Moldovan et al., 2015]

Les auteurs de [Kaur and Chana, 2014] ont proposé le Framework QoS-Aware Resource Elasticity (QRE) pour la gestion de l'élasticité des ressources au niveau application tout en considérant les exigences de qualité de service (QoS). L'approche proposée exprime les attributs de qualité de service d'une application (i.e., temps de réponse et taux d'utilisation de ressources) en des attributs de ressources Cloud afin de garantir leur élasticité. Le

Framework repose sur un modèle analytique permettant d'estimer les ressources requises pour satisfaire la charge de travail d'une application donnée, afin de garantir des exigences de qualité de service. Comme le montre la Figure 3.7, le Framework QRE comprend un composant Workload Analyzer qui interagit avec les utilisateurs d'une application et le composant QoS Mapper pour décider si une requête utilisateur est acceptée ou non. Le composant Application Centric Behavior Analyzer analyse le comportement de l'application afin de prédire la fréquence d'arrivée des différentes tâches par rapport à la variation du taux d'arrivée et la charge de travail. Le composant Resource Centric Behavior Analyzer récupère les informations sur le taux d'utilisation de ressources actuellement dans le système en utilisant le composant Moniteur propre à chaque fournisseur IaaS. Le composant QoS Mapper transforme les exigences de qualité de service de l'application à des allocations de ressources. Pour accomplir cette tâche, le composant QoS Mapper consulte la base de données Performance Database qui contient des informations sur la charge de travail actuelle et les attributs de qualité de service (QoS). Enfin, le composant Elasticity Controller reçoit des requêtes du composant QoS Mapper pour l'ajout et la suppression des instances de machines virtuelles sur l'infrastructure Cloud.

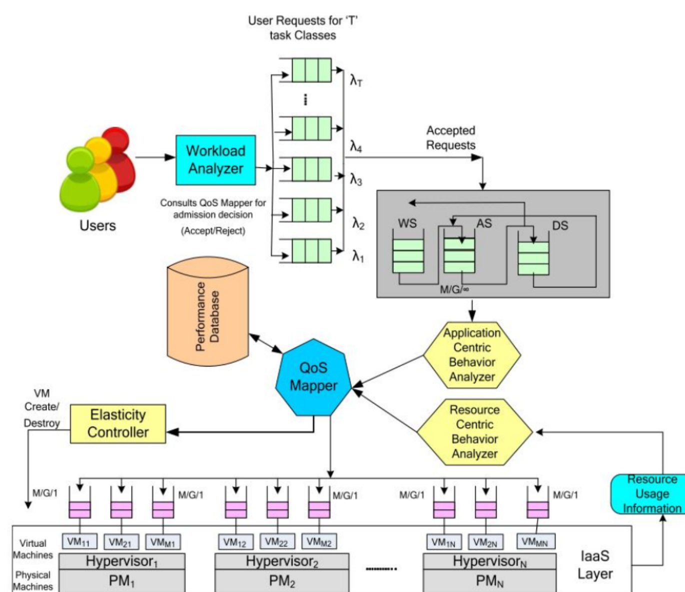


Figure 3.7 – Architecture du Framework QoS-Aware Resource Elasticity (QRE) [Kaur and Chana, 2014]

3.1.1 Synthèse

Les différentes approches présentées ici fournissent des mécanismes et des solutions pour la gestion de l'élasticité des systèmes Cloud. Ces travaux introduisent des modèles de contrôleur d'élasticité autonomique permettant d'analyser, de planifier et de gérer la consommation des ressources informatiques dans les systèmes Cloud au niveau des différentes couches de ces systèmes (infrastructure ou IaaS, plateforme ou PaaS et application ou SaaS). Les travaux cités reposent globalement sur une approche de développement basée sur une boucle de contrôle fermée pour la surveillance, l'analyse et l'application d'actions de reconfiguration selon les décisions prises. Les différentes tâches des processus d'adaptation implémentés reposent sur des outils modulaires fortement dépendants d'un

langage de programmation ou d'une technologie donnée, ce qui contraint fortement leur utilisation.

Les solutions mentionnées traitent un aspect particulier de l'élasticité. Précisément, certaines se focalisent sur l'élasticité horizontale (H) et la migration (M) tandis que d'autres traitent uniquement de l'élasticité verticale (V). Certaines se focalisent sur une ou plusieurs couches du Cloud (IaaS, PaaS, SaaS) et d'autres introduisent des stratégies d'élasticité uniquement à un seul niveau. Certains des environnements étudiés introduisent un modèle d'élasticité réactive (R) tandis que d'autres proposent un modèle d'élasticité prédictive (P). Enfin, le Tableau 3.1 résume l'analyse des travaux étudiés selon le modèle d'élasticité fourni, les couches du Cloud affectée ainsi que les méthodes d'élasticité fournies pour les solutions proposées.

TABLE 3.1 – Étude d'environnements pour la gestion autonomique de l'élasticité dans le Cloud

Solution	Modèle d'élasticité		Couche du Cloud			Méthode d'élasticité		
	R	P	IaaS	PaaS	SaaS	H	V	M
[Letondeur, 2014]	√	-	√	-	-	√	√	-
[Al-Dhuraibi, 2018]	√	-	√	√	-	√	√	√
[Dupont, 2016]	√	-	√	-	√	√	√	√
[Kranas et al., 2012]	√	√	-	√	√	√	√	-
[Loff and Garcia, 2014]	-	√	√	√	-	-	√	-
[Moldovan et al., 2015]	√	-	√	√	-	√	√	-
[Kaur and Chana, 2014]	√	-	√	-	√	-	-	-

3.2 Modèles formels pour l'élasticité dans le Cloud

Il existe de nombreux travaux de recherche dans la littérature traitant de la définition de modèles et approches formels pour la spécification et l'analyse des systèmes Cloud sur plusieurs aspects. Des travaux comme [Freitas et al., 2012, Uriarte et al., 2014, Bósa et al., 2015, Kikuchi and Hiraishi, 2014, Rady, 2013, Benzadri et al., 2013] ont abordé plusieurs notions liées aux systèmes Cloud telles que la qualité de service (QoS) et contrats de niveau de services (SLA), les interactions entre les systèmes Cloud et leurs clients, l'amélioration de la fiabilité des infrastructure Cloud, la disponibilité des services Cloud

ou encore la modélisation de la structure des services Cloud. Cependant, peu de travaux se sont intéressés aux comportements élastiques des systèmes Cloud. En effet, des fondations conceptuelles solides pour la modélisation et l'analyse des comportements élastiques du Cloud demeurent peu fréquentes. Dans ce contexte, quelques travaux ont proposé des approches à cadre formel pour la modélisation et l'analyse des comportements des systèmes Cloud élastiques. Dans cette Section, nous présentons et discutons certains de ces travaux, reposant sur des formalismes différents.

Les auteurs de [Bersani et al., 2014] ont proposé une approche de modélisation d'un nombre de concepts et propriétés relatives aux comportements élastiques des systèmes Cloud. Les auteurs se basent sur le formalisme CLTLt(d) (*Timed Constraint Linear Temporal Logic*). Précisément, les auteurs proposent l'utilisation de la logique temporelle CLTLt(d) pour la définition de certaines propriétés inhérentes à l'élasticité, la gestion de ressources et la qualité de service (QoS) dans les systèmes Cloud. L'utilisation de cette logique temporelle permet de recourir à des outils de vérification pour s'assurer de la satisfaction ou, au contraire, de la violation des propriétés introduites lors de l'exécution d'un système Cloud élastique. En termes de modélisation, l'architecture des systèmes Cloud est abstraite pour n'adresser que les ressources au niveaux infrastructure. Précisément à travers le nombre de machines virtuelles. De ce fait, les auteurs décrivent des mécanismes d'élasticité horizontale uniquement au niveau infrastructure, pour l'ajout et le retrait (scale-out/in) de machines virtuelles. L'approche proposée est validée à l'aide d'un outil hors-ligne basé sur des solveurs SAT et SMT. Cet outil analyse les traces d'exécution du système résultant de simulations effectuées en ligne. Durant les simulations conduites, le système Cloud est confronté à des charge de travail en entrée (workload) différentes afin de déclencher et d'observer son comportement élastique.

Les auteurs de [Yataghene et al., 2014] ont introduit un modèle analytique basé sur la théorie des files d'attente (TFA). Ils proposent un modèle de files d'attente (MFA) avec nombre variable de serveurs. Ils définissent des stratégies horizontales au niveau application, pour l'adaptation (scaleout/in) aux variations de la charge de travail des processus métier basés service (service-based business processes SBP). Dans cette approche, les auteurs modélisent la charge de travail en entrée (workload) du système à travers un processus de Poisson suivant un taux d'arrivée des requêtes λ. Le nombre de requêtes traitées (quittant le système) suit une loi exponentielle selon le taux μ. En outre, le service basé Cloud est modélisé par une chaine de Markov (CdM). Cette modélisation décrit l'état du système à travers la taille des files d'attente pour chaque instance de service déployée, comme le montre la Figure 3.8. Ainsi, des métriques telles que le nombre de serveurs et le temps de réponse moyen sont obtenues à l'aide de calculs de formules probabilistes. En outre, les auteurs fournissent une évaluation quantitative des comportements définis par le biais de scénarios simulés afin de valider leur approche. Toutefois, aucune vérification formelle n'est fournie pour les comportements définis.

Les auteurs de [Amziani, 2015] ont proposé un Framework formel basé sur les réseaux de Petri (*Petri nets : PN / Colored Petri nets : CPN*) pour la modélisation de l'élasticité des processus métier basés service (SBP) dans le Cloud. Dans ces travaux, les auteurs se focalisent sur la modélisation de l'élasticité au niveau application (SaaS) et les détails des autres couches d'une architecture Cloud ne sont pas pris en charge par leur modèle. De ce fait, l'approche proposée se concentre sur l'élasticité horizontale au niveau service tout en considérant les notions de charge de travail et de temps de réponse afin d'adresser la qualité de service du système. Les auteurs décrivent un contrôleur d'élasticité, basé sur le modèle MAPE-K et mettent en œuvre des stratégies d'élasticité pour la duplication

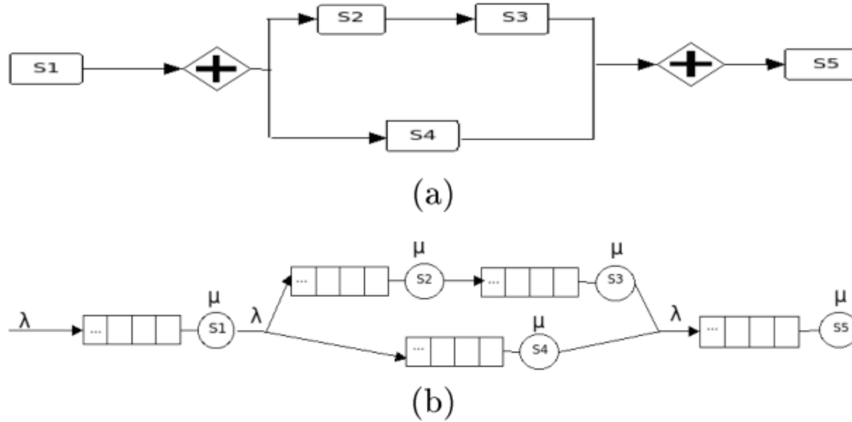


FIGURE 3.8 – Principe de modélisation à base de files d'attente [Yataghene et al., 2014]

(scale-out), la consolidation (scale-in) ainsi que pour le routage (routing) des ressources Cloud au niveau service (application). Les stratégies définies sont comparées en termes de fiabilité, performances et de taux de consommation des ressources. En outre, l'approche proposée est vérifiée via une technique basée sur la vérification de preuves. Les auteurs utilisent SNAKES, un outil basé sur les réseaux de Petri qui vérifie l'accessibilité de graphes. Cet outil vérifie le bon fonctionnement des stratégies définie qui sont simulées à la phase de conception. La Figure 3.9 donne une vue d'ensemble de la boucle du contrôleur d'élasticité pour les SBP.

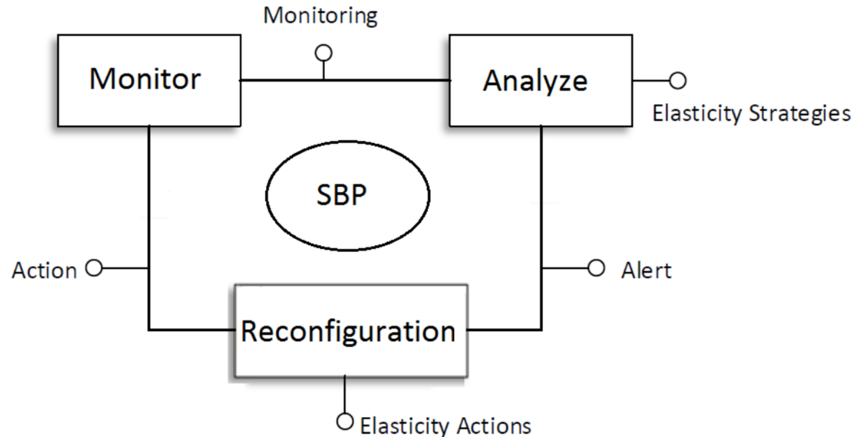


FIGURE 3.9 – Vue d'ensemble du contrôleur d'élasticité pour les SBP [Amziani, 2015]

Les trois approches citées jusqu'ici sont basées sur la logique CLTLt(d) [Bersani et al., 2014], sur les chaines de Markov [Yataghene et al., 2014] ou encore sur les réseaux de Petri [Amziani, 2015]. Ces approches considèrent les systèmes Cloud à un très haut niveau d'abstraction. À travers les formalismes utilisés, les travaux mentionnés ne considèrent respectivement que le nombre de machines virtuelles, la taille des files d'attente et le nombre de requêtes comme principales variables définissant l'état du système Cloud, et impactant l'aspect décisionnel relatif à l'élasticité. De ce fait, [Yataghene et al., 2014, Amziani, 2015] introduisent des stratégies horizontales ne s'appliquant qu'au niveau application d'un système Cloud, et [Bersani et al., 2014] ne l'adresse qu'au niveau infrastructure. De plus, ces travaux ne fournissent pas de support pour l'exécution autonomique des com-

portements élastiques introduits.

Les auteurs de [Sahli, 2017] ont proposé une approche formelle basée sur les systèmes réactifs bigraphiques (Bigraphical Reactive Systems : BRS) pour la modélisation des aspects tant structurels que comportementaux des systèmes Cloud élastiques. Les auteurs ont proposé une sémantique bigraphique et une logique de typage afin de proposer une modélisation robuste et générique des architectures basées Cloud. Dans ces travaux, les auteurs définissent un nombre de règles de réaction bigraphiques afin de représenter les interactions entre un système Cloud et ses clients (utilisateurs, développeurs). Les méthodes d'élasticité horizontale, verticale et migration sont également représentée par le biais de règles de réaction bigraphiques s'appliquant en multi-couches complètes, c'est-à-dire aux niveaux infrastructure, plateforme et application d'un système Cloud. En outre, les auteurs présentent MoVeElastic, un Framework permettant d'implémenter les spécifications bigraphiques à l'aide du langage de spécification formelle Maude. Ce Framework permet de vérifier formellement les comportements élastiques définis à travers une méthode de model-checking basé sur LTL et via une technique basée sur la vérification par invariants. La Figure 3.10 montre le principe de fonctionnement de MoVeElastic.

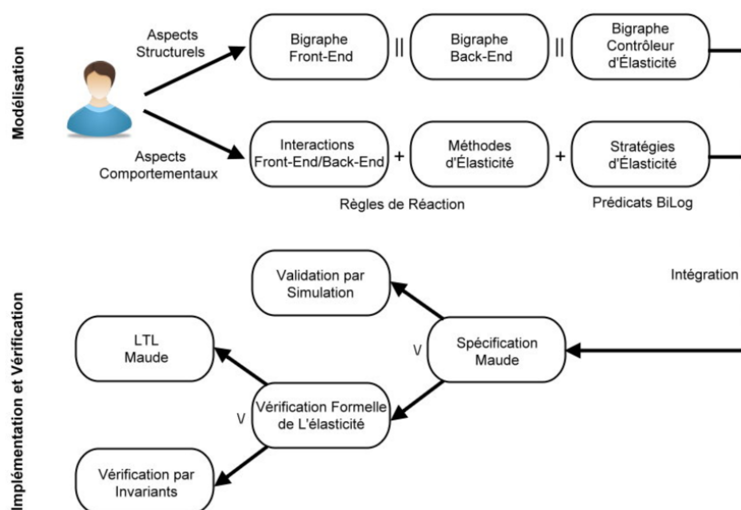


FIGURE 3.10 – Fonctionnement de MoVeElastic [Sahli, 2017]

À travers leurs travaux, les auteurs de [Sahli, 2017] ont adressé un certain nombre de limitations identifiées au niveau des trois premiers travaux ntroduits. En effet, l'approche de modélisation formelle à base des BRS permet de modéliser un système Cloud de manière générique et générale en tenant compte de sa structure multi-couches aux niveaux infrastructure, plateforme et application. De plus, les auteurs définissent des mécanismes afin de gérer l'élasticité horizontale, verticale et la migration dans les systèmes Cloud modélisés. Néanmoins, à l'instar des travaux précédemment mentionnés, les auteurs de [Sahli, 2017] ne fournissent pas de support pour l'exécution autonomique des comportements élastiques définis. De plus, ils ne définissent pas de stratégies d'élasticité autonomiques. Les auteurs recourent au langage de stratégies du système Maude afin de guider les reconfigurations des systèmes Cloud modélisés. Cela revient à imposer un ordre de déclenchement des différents règles définies en vue d'atteindre une configuration cible désirée, selon la méthode d'élasticité déclenchée. En outre, les auteurs ne proposent aucune évaluation des comportements introduits sur le plan quantitatif (performances, coûts).

3.2.1 Synthèse

Afin de synthétiser et discuter les apports et manques des différents travaux cités, nous identifions quelques critères et caractéristiques qui nous ont servis de base pour l'étude critique des travaux tentant d'adresser la problématique de la spécification formelle de l'élasticité du Cloud. À savoir, le formalisme utilisé, la modélisation de la structure des systèmes Cloud, les couches (infrastructure, application) ciblées par les comportements élastiques introduits, les méthodes d'élasticité fournies (horizontale, verticale, migration, en plus du load balancing), ainsi que la prise en charge stratégies d'élasticité. Du point de vue de la vérification et de la validation des approches introduites, nous mettons l'accent sur la technique de vérification formelle utilisée, ainsi que la démarche suivie pour l'évaluation et la validation quantitative. La Table 3.2 résume l'analyse de ces critères au niveau des travaux étudiés.

Niveaux de détails

Dans ce contexte, nous observons que les approches étudiées proposent des solutions très différentes mais complémentaires. Du point de vue de la spécification des systèmes Cloud et de leurs comportements élastiques, seuls [Sahli, 2017] adressent les structures du Cloud de manière générique et détaillée et en multi-couches (càd. aux niveaux infrastructure, plateforme et application). Les autres travaux comme [Yataghene et al., 2014, Amziani, 2015] abordent le niveau service du Cloud mais en abstrayant complètement les architectures sous-jacentes. Réciproquement, [Bersani et al., 2014] décrit l'infrastructure du Cloud mais cette fois également à un niveau d'abstraction considérable, en traitant des machines virtuelles sans aborder les détails de l'architecture du Cloud. Du point de vue de la structure en couches du Cloud affectée par l'élasticité, les travaux cités s'intéressent uniquement aux niveaux qu'ils considèrent dans leurs modèles. De ce fait, [Sahli, 2017] se démarque en proposant des comportements élastiques en multi-couches. De plus, tous les travaux mentionnés ne fournissent que des comportements d'élasticité horizontale, à l'exception de [Amziani, 2015] qui évoque le routage des requêtes et de [Sahli, 2017] qui abordent également l'élasticité verticale, la migration et le load-balancing.

Stratégies d'élasticité et états du système

Les auteurs de [Sahli, 2017] couvrent l'essentiel des opérations d'adaptation possibles à tous les niveaux du Cloud de manière générique, fournissant ainsi une méthode de référence pour la gestion de l'élasticité. Cependant, les auteurs ne proposent pas de stratégies pour la gestion autonomique des comportements élastiques introduits. De plus, ils ne proposent pas d'approche expérimentale afin d'étudier et de valider leurs approches d'un point de vue quantitatif. Dans ce contexte, les autres travaux proposent des stratégies d'élasticité horizontale au niveau service [Yataghene et al., 2014, Amziani, 2015]et au niveau infrastructure [Bersani et al., 2014]. Les stratégies proposées sont de nature réactives (i.e., de la forme Si condition Alors action) et introduisent des mécanismes pour l'ajout et le retrait des ressources de manière autonomique. Notons que [Amziani, 2015] décrivent également des stratégies proactives (prédictives) pour la gestion de l'élasticité. Les stratégies définies fonctionnent sur un principe de seuils (e.g. nombre maximal de requêtes tolérées, nombre maximal/minimal de ressources déployées) afin de gérer l'élasticité. En raison de l'important niveau d'abstraction des modélisations fournies, la représentation des états de sur/sous-dimensionnement du système reposent sur

TABLE 3.2 – Étude de modèles formels pour la spécification et la vérification de l'élasticité dans le Cloud

Approche		[Bersani et al., 2014]	[Amziani, 2015]	[Yataghene et al., 2014]	[Sahli, 2017]
Modélisation de l'architecture du Cloud		-	-	-	√
Méthode d'élasticité	Horizontale	√	√	√	√
	Verticale	-	-	-	√
	Migration	-	-	-	√
	Load Balancing	-	√	-	-
Couche du Cloud	Infrastructure	√	-	-	√
	Plateforme	-	-	-	√
	Application	-	√	√	√
Stratégies d'élasticité	Réactive	√	√	√	-
	Proactive	√	-	-	-
Formalisme / modèle formel		CLTLt(d)	PN/CPN	CdM/TFA	BRS/-Maude
Technique de vérification formelle		Solveurs SAT/SMT	Vérification de preuve	-	Model-Checking / Invariants
Évaluation quantitative	Technique	Simulation	Simulation	Calcul de probabilités / MFA	-
	Support outillé	-	-	-	-

des considérations très simples comme le nombre de requêtes dans la file d'attente [Yataghene et al., 2014] ou encore le nombre de services/VMs déployés [Amziani, 2015, Bersani et al., 2014].

Au meilleur de nos connaissances, l'ensemble des travaux cités ne permettent pas de décrire un état complexe et multifactoriel d'un système Cloud. À savoir, en prenant

en compte la charge de travail en entrée, le nombre de service déployés, le nombre de machines virtuelles déployées, la quantité de ressources disponibles au niveau des VMs ou encore la quantité des ressources disponibles au niveau des serveurs physiques.

Évaluation quantitative de l'élasticité

Du point de vue de l'évaluation et la validation quantitative des comportements élastiques introduits, [Amziani, 2015, Bersani et al., 2014, Yataghene et al., 2014] proposent une approche à base de simulations afin d'observer et de mesurer les performances induites par les comportements élastiques introduits. Les auteurs de [Amziani, 2015] procèdent à des simulations au niveau de la conception. Dans [Yataghene et al., 2014], les auteurs fournissent de simulations à base de files d'attentes où l'arrivée et le départ des requêtes dans le système sont représentés par des processus de vie et de mort (life and death process). Le nombre de ressources nécessaires est calculé à l'aide de formules probabilistes et le système n'est pas confronté à des variations dans la charge de travail. Dans [Bersani et al., 2014], les auteurs simulent le fonctionnement d'un service Cloud en ligne puis analysent les traces d'exécution obtenues. Le système est confronté à plusieurs modèles de charges de travail afin d'observer son comportement. Dans ces trois travaux, les approches de validation quantitative proposées sont très spécifiques et sont très difficilement généralisable pour tout système Cloud. De plus, les auteurs ne fournissent pas de support logiciel pour la simulation de l'élasticité multi-couches. Quant à [Sahli, 2017], les auteurs ne proposent pas de support pour l'évaluation quantitative de l'élasticité.

3.3 Conclusion

Dans ce chapitre, nous avons présenté plusieurs travaux liés à l'élasticité dans le Cloud, existants dans la littérature. Dans un premier temps, nous avons étudié quelques environnements conçus pour la gestion autonomique de l'élasticité. Dans un deuxième temps, nous avons étudié quelques approches et modèles formels ayant proposé des solutions pour la spécification et la vérification de l'élasticité dans le Cloud.

Nous avons étudié l'ensemble des solutions et approches présentées selon un certain nombres de critères, que nous avons jugé pertinents, afin de dégager leurs apports et manques, dans le but de préparer le lecteur à bien situer nos contributions.

Chapitre 4

Prérequis et fondements formels

Sommaire

4.1 Introduction 48
4.2 Les Systèmes Réactifs Bigraphiques 49
4.2.1 Anatomie et forme graphique des bigraphes 50
4.2.2 Définitions Formelles 52
4.2.3 Opérations sur les bigraphes 53
4.2.4 Forme algébrique 56
4.2.5 Logique de typage (*sorting*) 59
4.2.6 Dynamique des bigraphes 60
4.2.7 Bigraphes concrets et bigraphes abstraits 61
4.2.8 Outils pratiques autour des BRS 62
4.3 Langage Maude 63
4.3.1 Syntaxe et notations 63
4.3.2 Modules fonctionnels 64
4.3.3 Modules systèmes 65
4.3.4 Vérification formelle dans Maude 67
4.4 Théorie des files d'attente 68
4.4.1 Objectif de l'analyse des files d'attente 69
4.4.2 Caractéristiques des systèmes de files d'attente 70
4.4.3 Modèles de files d'attente 71
4.4.4 Théorie des files d'attente et gestion dynamique des ressources . . . 72
4.5 Conclusion 73

4.1 Introduction

Avec l'évolution des technologies de l'information et de la communication, les systèmes informatiques ont considérablement gagné en complexité. Leur conception est devenue de plus en plus difficile à maîtriser, et leur maintenance, de plus en plus coûteuse à assurer. Les méthodes de développement classiques, bien que coûteuses, se montrent souvent inefficaces pour garantir de manière exhaustive le champ de validité d'un système, et peinent à assurer sa fiabilité et robustesse. De par le développement du génie logiciel et l'émergence d'un bon nombre de concepts d'informatique théorique, une des solutions proposées pour spécifier et analyser les systèmes informatiques complexes consiste à recourir aux méthodes formelles.

Les méthodes formelles sont des méthodes outillées pour la spécification et l'analyse de systèmes au sens large. Elles reposent sur des fondements mathématiques pour offrir des mécanismes d'abstraction et de modélisation non ambigus, ainsi qu'une rigueur et une précision considérables dans la spécification des aspects structurels et comportementaux des systèmes, ce qui permet de réduire et maitriser leur complexité. Ces méthodes permettent de décrire, de manière précise, des propriétés intrinsèques de sûreté et de vivacité ainsi que des propriétés relatives au fonctionnement des systèmes conçus. Les méthodes formelles fournissent particulièrement un moyen de vérifier et garantir la satisfaction de ces propriétés de manière systématique, ce qui réduit considérablement les coûts de maintenance (principalement corrective) de ces systèmes.

Dans ce Chapitre, nous présentons les principaux modèles et théories formelles utilisées dans ce manuscrit afin de préparer le lecteur à une bonne compréhension des contributions qui y sont présentées. La Section (4.2) présente le formalisme des systèmes réactifs bigraphiques (BRS). Nous y présentons les principales définitions liées à ce modèle formel, ainsi que les mécanismes qu'il offre pour la spécification des systèmes sur le plan structurel et comportemental. La Section (4.3) s'intéresse au langage de spécification formelle Maude. Nous présentons les principales caractéristiques de ce langage, ses définitions liées et expliquons son fonctionnement, notamment pour l'implémentation, l'exécution et la vérification des propriétés liées au fonctionnement d'un système donné. Le langage Maude permet d'encoder les spécifications bigraphiques de manière à préserver leur sémantique d'une part tout en permettant de les enrichir d'un autre. L'association des BRS et de Maude permet, d'un côté, d'apporter une formalisation forte et robuste d'un système tout en décrivant son comportement dynamique intrinsèque. D'un autre côté, les outils d'analyse formelle de Maude permettent d'assurer une conception correcte de ces comportements. La Section (4.4) traite de la théorie des files d'attente. Nous présentons la philosophie de cette théorie et expliquons son utilisation, pour la prise en charge des mesures de performances des systèmes. La théorie des files d'attente fournit un support mathématique pour l'évaluation quantitative des systèmes. Si l'association des BRS et de Maude permet de concevoir et de vérifier le bon fonctionnement d'un système, la théorie des files d'attente vient compléter cette association en introduisant des mécanismes pour l'évaluation et la validation des systèmes conçus, notamment en termes de coûts et de performances. Enfin, nous concluons ce chapitre par la Section 4.5.

4.2 Les Systèmes Réactifs Bigraphiques

Les systèmes réactifs bigraphiques (SRB) ou *Bigraphical Reactive Systems* (BRS) sont un formalisme conçu pour la modélisation de l'évolution temporelle et spatiale du calcul. La théorie de bigraphes a récemment été introduite par Robin Milner [Milner, 2008] en vue de fournir un modèle graphique intuitif à même de représenter la localité et la connectivité des systèmes dits ubiquitaires. Un système bigraphique réactif est constitué d'un ensemble de bigraphes représentant l'état du système et un ensemble de règles de réaction décrivant son évolution. La théorie des bigraphes a été développée avec deux objectifs principaux : (1) être en mesure d'intégrer dans le même formalisme les aspects importants des systèmes ubiquitaires ; et (2) de fournir une unification des théories existantes en développant une théorie générale, dans laquelle les différents calculs existants pour la concurrence et la mobilité, tels que le calcul des systèmes communicants [Milner, 1980], le pi-calcul [Milner et al., 1992], le calcul ambiant [Milner, 1993] et les réseaux de Pétri, peuvent être représentés avec une théorie comportementale uniforme. Cette dernière est obtenue en

représentant la dynamique des bigraphes par une définition abstraite de règles de réaction à partir de laquelle un système de transition peut être dérivé pour décrire le comportement des systèmes modélisés. Dans cette Section, nous introduisons les bigraphes d'une manière informelle avant d'aborder leur définition formelle et les différentes notions qui y sont liées. Le contenu de ce Chapitre est tiré des différents ouvrages (livres, publications et rapports techniques) publiés par Robin Milner [Milner, 2001a, Milner, 2001b, Milner, 2005, Milner, 2006, Birkedal et al., 2007, Krivine et al., 2008, Milner, 2008, Milner, 2009]. Dans le cadre de notre apprentissage des bigraphes, les définitions apportées ici correspondent à celles initialement posées pour les bigraphes classiques.

4.2.1 Anatomie et forme graphique des bigraphes

Prenons l'exemple d'un système représenté par un bigraphe B comme montré dans la figure 4.1. Dans la forme graphique, les composants ou entités (physiques ou logiques) qui constituent le système sont exprimés par des nœuds pouvant prendre différentes formes géométriques (ovale, carré, rectangle, etc.). Les nœuds d'un bigraphe possèdent un type donné, appelé contrôle et désigné par un court identifiant alphabétique (e.g. A, B, EC, etc.). L'ensemble des contrôles d'un bigraphe constitue la signature de ce dernier. La répartition spatiale des nœuds est exprimée sous forme d'imbrications hiérarchiques entre les différents nœuds du système. Un nœud est dit atomique s'il ne peut contenir d'autres nœuds. Les interactions entre les nœuds sont exprimées par des connexions non-binaires sous forme de liens (reliant deux entités ou plus) appelés Hyper-arcs. Ces connexions peuvent exprimer des canaux de communication entre les nœuds, ainsi que d'autres relations abstraites les liant. Un nœud peut posséder zéro ou plusieurs ports, représentés par des puces rondes sur sa membrane indiquant les connexions possibles. Les nœuds du même contrôle possèdent le même nombre de ports. Les rectangles en pointillés représentent des régions, aussi appelées racines, décrivant des parties distinctes du système. Les carrés de couleur grise, appelés sites, représentent des parties abstraites du système non considérées par le modèle. Les régions et les sites sont numérotés par des entiers naturels (à partir de 0). En plus des hyper-arcs, un bigraphe peut posséder d'autres types de liens de communications : les noms internes et les noms externes. Ils expriment de potentiels liens avec d'autres bigraphes qui représentent des environnements externes.

Un bigraphe dispose d'une interface indiquant ses possibilités d'interactions avec son environnement extérieur. Par exemple, le bigraphe B a deux sites, deux régions et deux ensembles de noms externes $\{y_0, y_1, y_2\}$ et noms internes $\{x_0, x_1\}$. La paire $< 2, \{x_0, x_1\} >$ désigne l'interface interne de B, alors que $< 2, \{y_0, y_1, y_2\} >$ représente son interface externe. Finalement, un bigraphe peut être exprimé en termes de localité à travers la distribution spatiale des nœuds, alors que sa connectivité est exprimée par des liens. Ces deux représentations définissent deux graphes distincts. En effet, un bigraphe peut être vu comme une combinaison de deux structures indépendantes : le graphe de places et le graphe de liens, d'où le préfixe « bi » [Milner, 2008]. L'intersection entre ces deux graphes est un ensemble commun de nœuds, correspondant aux entités réelles ou virtuelles du bigraphe. Le graphe de places prend la structure d'une forêt, représentant la distribution spatiale des différentes entités en ignorant leurs connexions. Le graphe de liens est un hypergraphe donnant le réseau de connectivité des différents nœuds en ignorant leur localité. Alors qu'un arc dans le graphe de places montre la relation d'imbrication entre les éléments de l'application, un hyper-arc dans le graphe de liens établit une connexion entre les ports de ces éléments et peut également relier ses noms internes et externes.

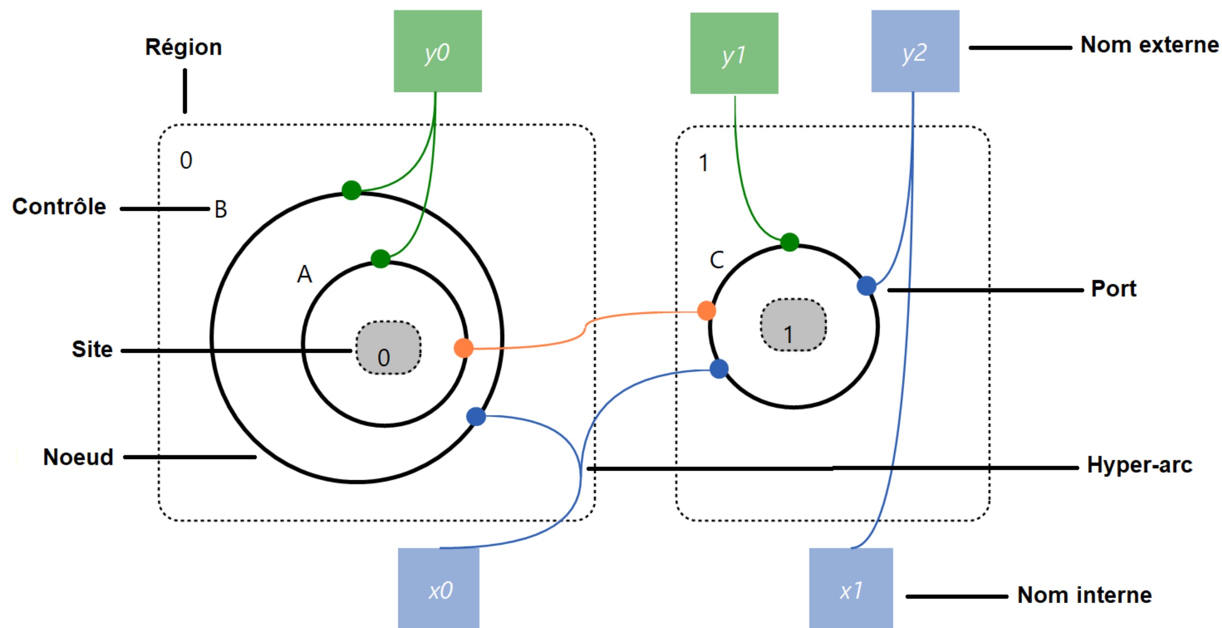


Figure 4.1 – Anatomie des bigraphes

Chaque arbre dans le graphe de places représente une région (dont elle est la racine) qui peut contenir des nœuds et des sites, correspondant aux feuilles de l'arbre.

La Figure 4.2 illustre le graphe de places du bigraphe B (illustré en Figure 4.1).

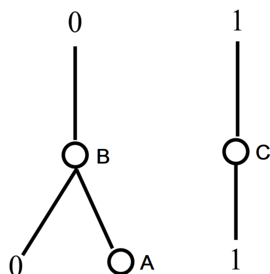


Figure 4.2 – Graphe de places

La Figure 4.3 illustre le graphe de liens du bigraphe B.

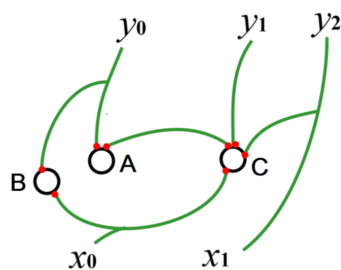


Figure 4.3 – Graphe de liens

4.2.2 Définitions Formelles

Par convention avec la définition des bigraphes classiques [Milner, 2008], nous écrivons $S \uplus S'$ pour désigner l'union de deux ensembles S et S' reconnus ou supposés disjoints ($S \bigcap S' = \emptyset$). Si la fonction f a un domaine S et $S' \subseteq S$, alors $f \restriction S'$ désigne la restriction de f vers S'. Pour deux fonctions f et g ayant des domaines disjoints S et T, écrivons $f \uplus g$ pour la fonction ayant le domaine $S \uplus T$ de telle sorte que $(f \uplus g) \restriction S = f$ et $(f \uplus g) \restriction T = g$. Nous considérons un entier naturel k comme un ordinal fini $k = 0, 1, \ldots, k-1$.

Définition 3 (signature) On note $(\mathcal{K}, ar)$ la signature d'un bigraphe B. Elle comporte un ensemble $\mathcal{K}$ de contrôles et une fonction de transformation $ar : \mathcal{K} \rightarrow N$, qui associe une arité à chaque contrôle. Appliquée aux nœuds de B, la signature $\mathcal{K}$ assigne à chaque nœud un contrôle dont l'arité indique le nombre de ports. Une signature du bigraphe B, précédemment présenté dans la Figure 4.1, est donnée par $\mathcal{K} = \{A : 2, B : 2, C : 4\}$.

Définition 4 (bigraphe) Formellement, un bigraphe B prend la forme suivante :

$$B = (V_B, E_B, ctrl_B, B^P, B^L) : I_B \rightarrow J_B$$

- V_B est un ensemble fini de nœuds.
- E_B est un ensemble fini d'hyper-arcs.
- $ctrl_B : V_B \rightarrow \mathcal{K}$ est une fonction de transformation qui associe à chaque nœud $v_i \in V_B$ un contrôle $c \in \mathcal{K}$ indiquant le nombre de ports. La signature $\mathcal{K}$ est un ensemble fini de contrôles.
- $B^P = (V_B, ctrl_B, prnt_B) : m \rightarrow n$ est le graphe de places associé à B. m et n représentent le nombre de sites et le nombre de régions.$prnt_B : m \uplus V_B \rightarrow V_B \uplus n$ est une fonction de parentalité qui associe à chaque nœud ou site son parent hiérarchique.
- $B^L = (V_B, E_B, ctrl_B, link_B) : X \rightarrow Y$ est le graphe de liens de B, où $link_B : X \uplus P \rightarrow E_B \uplus Y$ est une fonction de transformation. X, Y et P représente respectivement l'ensemble de noms internes, l'ensemble de noms externes et l'ensemble de ports de B.
- $I = < m, X >$ et $J = < n, Y >$ représentent respectivement les interfaces internes et externes du bigraphe B.

Définition 5 (graphe de places) . Le graphe de places est défini formellement par :

$$B^P = (V_B, ctrl_B, prnt_B) : m \rightarrow n$$

- V_B est un ensemble fini de nœuds.
- $ctrl_B$ est la fonction de contrôle.
- $prnt_B : m \uplus V_B \rightarrow V_B \uplus n$ est une fonction de parentalité. La notation $m \uplus V_B$ indique que les deux ensembles restent disjoints.
- La notation $m \rightarrow n$ désigne l'interface interne (m) et l'interface externe (n) du graphe de place B^P.

Définition 6 (graphe de liens) . Le graphe de liens est défini formellement par :

$$B^L = (V_B, E_B, ctrl_B, link_B) : X \rightarrow Y$$

— V_B est un ensemble fini de nœuds.
— E_B est un ensemble fini d'hyper-arcs.
— $ctrl_B$ est la fonction de contrôle.
— $link_B : X \uplus P \to E_B \uplus Y$ est une fonction de transformation qui montre la connectivité des noms internes X ou des ports P avec les noms externes Y ou les hyper-arcs E.

4.2.3 Opérations sur les bigraphes

Dans cette Section, nous montrons comment des bigraphes complexes peuvent être construits à partir de bigraphes élémentaires, en appliquant les opérations de composition et de produit tensoriel [Milner, 2008]. Nous fournissons les définitions formelles de ces deux opérations bigraphiques et nous expliquons leur principe à travers des exemples.

Composition

L'opération de composition consiste à placer un bigraphe dans un contexte représenté par un autre bigraphe. La composition d'un bigraphe F dans un bigraphe G, notée $G \circ F$, est applicable si et seulement si l'interface externe de F correspond à l'interface interne de G. La composition de F dans G se fait en hébergeant les régions de F dans les sites de G et en fusionnant les noms externes de F avec les liens de G qui ont des noms internes correspondants. Dans ce qui suit, nous donnons la définition formelle de l'opération de composition de deux bigraphes (F et G). Pour cela, nous définissons d'abord la composition au niveau du graphe de places (F^P et G^P) et graphe de liens (F^L et G^L) avant de présenter la composition au niveau du bigraphe.

Définition 7 (composition de deux graphes de places) . Si $F^P : k \to m$ et $G^P : m \to n$ sont deux graphes de places avec des supports disjoints, la composition de F^P dans G^P ($G^P \circ F^P$) :

$$G^P \circ F^P = (V, ctrl, prnt) : k \to n \text{ (Figure 4.4)}$$

Donne l'ensemble de nœuds $V = V_{F^P} \uplus V_{G^P}$ et la fonction de contrôle $ctrl = ctrl_{F^P} \uplus ctrl_{G^P}$. Sa fonction de parentalité $prnt$ est définie par : Si $w \in k \uplus V$ est un site ou un nœud dans $G^P \circ F^P$ alors :

$$prnt(w) \stackrel{\text{def}}{=} \begin{cases} prnt_{F^P}(w) & \text{si } w \in k \uplus V_{F^P} \text{ et } prnt_{F^P}(w) \in V_{F^P}, \\ prnt_{G^P}(j) & \text{si } w \in k \uplus V_{F^P} \text{ et } prnt_{F^P}(w) = j \in m, \\ prnt_{G^P}(w) & \text{si } w \in V_{G^P}. \end{cases}$$

Définition 8 (composition de deux graphes de liens) Si $F^L : X \to Y$ et $G^L : Y \to Z$ sont deux graphes de liens avec des supports disjoints, la composition de F^L dans G^L ($G^L \circ F^L$) :

$$G^L \circ F^L = (V, E, ctrl, link) : X \to Z \text{ (Figure 4.5)}$$

Donne un ensemble de nœuds $V = V_{F^L} \uplus V_{G^L}$, un ensemble d'hyper-arcs $E = E_{F^L} \uplus E_{G^L}$ et une fonction de contrôle $ctrl = ctrl_{F^L} \uplus ctrl_{G^L}$. Sa fonction $link$ est définie par : Si $q \in X \uplus P_{F^L} \uplus P_{G^L}$ est un point de $G^L \circ F^L$ alors :

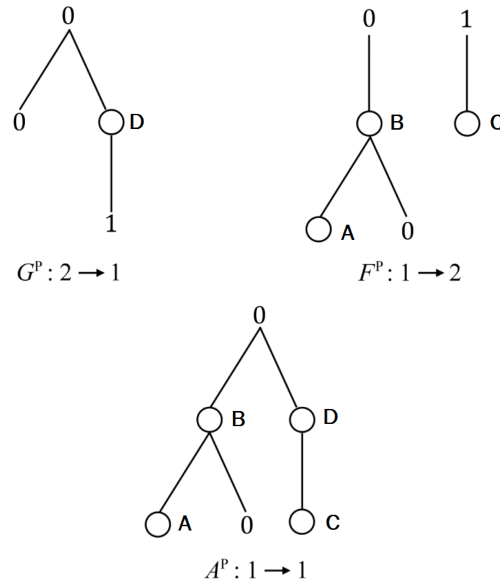


FIGURE 4.4 – Composition de deux graphes de places : $A^P = G^P \circ F^P$

$$link(q) \stackrel{\text{def}}{=} \begin{cases} link_{F^L}(q) & \text{si } q \in X \uplus P_{F^L} \text{ et } link_{F^L}(q) \in E_{F^L}, \\ link_{G^L}(y) & \text{si } q \in X \uplus P_{F^L} \text{ et } link_{F^L}(q) = y \in Y, \\ link_{G^L}(q) & \text{si } q \in P_{G^L}. \end{cases}$$

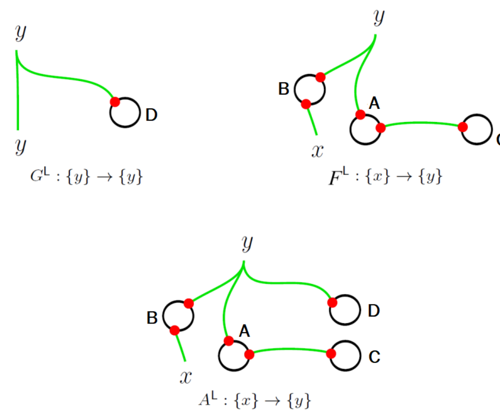


FIGURE 4.5 – Composition de deux graphes de liens : $A^L = G^L \circ F^L$

Définition 9 (composition de deux bigraphes) . Si $F :< k, X > \to < m, Y >$ et $G :< m, Y > \to < n, Z >$ sont deux bigraphes avec des supports disjoints, la composition de F dans G est donnée par :

$$G \circ F \stackrel{\text{def}}{=} (G^P \circ F^P, G^L \circ F^L) :< k, X > \to < n, Z >$$

L'exemple suivant (Figure 4.6) montre le bigraphe $A = G \circ F :< 1, x > \to < 1, y >$ résultant de la composition de deux bigraphes $G :< 2, y > \to < 1, y >$ et $F :< 1, x > \to < 2, y >$ dont les graphes de places et les graphes de liens sont composés dans les Figures 4.4 et 4.5 respectivement.

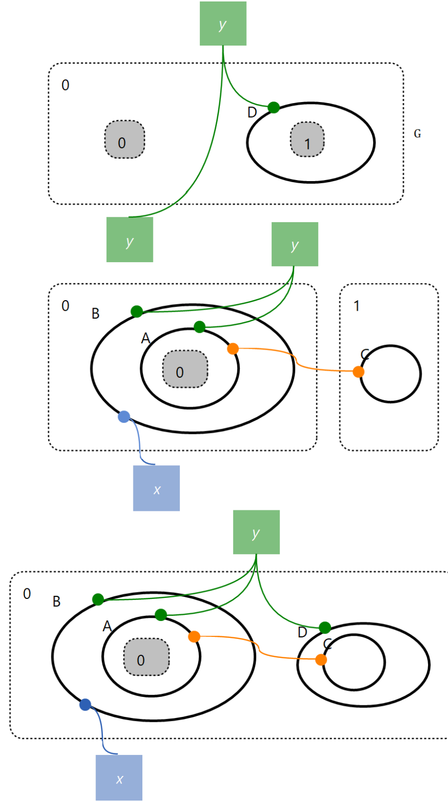


FIGURE 4.6 – Composition de deux bigraphes : $A = G \circ F$

Produit Tensoriel

Le produit tensoriel est un autre moyen de composer les bigraphes. Il consiste en la juxtaposition des régions de bigraphes en joignant les liens ouverts communs. Cette opération est également appelée composition horizontale. Comme pour la composition, nous définissons formellement le produit tensoriel au niveau des graphes de places et graphes de liens avant de définir cette opération au niveau de bigraphes.

Définition 10 (Produit tensoriel des graphes de places) . Si $B_i^P = (V_i, ctrl_i, prnt_i) : m_i \rightarrow n_i (i = 0, 1)$ sont deux graphes de places disjoints, leur produit tensoriel $B_0^P \otimes B_1^P$: $m_o + m_1 \rightarrow n_0 + n_1$ est donné par :

$$B_0^P \otimes B_1^P \stackrel{\text{def}}{=} (V_0 \uplus V_1, ctrl_0 \uplus ctrl_1, prnt_0 \uplus prnt'_1) \text{ (Figure 4.7)}$$

Où $prnt'_1(m_0 + i) = n_0 + j$ lorsque $prnt_1(i) = j$.

Définition 11 (Produit tensoriel des graphes de liens) . Si $B_i^L = (V_i, E_i, ctrl_i, prnt_i)$: $X_i \rightarrow Y_i (i = 0, 1)$ sont deux graphes de liens disjoints, leur produit tensoriel $B_0^L \otimes B_1^L$: $X_o \uplus X_1 \rightarrow Y_0 \uplus Y_1$ est donné par :

$$B_0^L \otimes B_1^L \stackrel{\text{def}}{=} (V_0 \uplus V_1, E_0 \uplus E_1, ctrl_0 \uplus ctrl_1, link_0 \uplus link_1) \text{ (Figure 4.8)}$$

Définition 12 (Produit tensoriel des bigraphes) . Si $B_i :< m_i, X_i > \rightarrow < n_i, Y_i >, (i = 0, 1)$ sont deux bigraphes avec des supports disjoints, leur produit tensoriel est donné

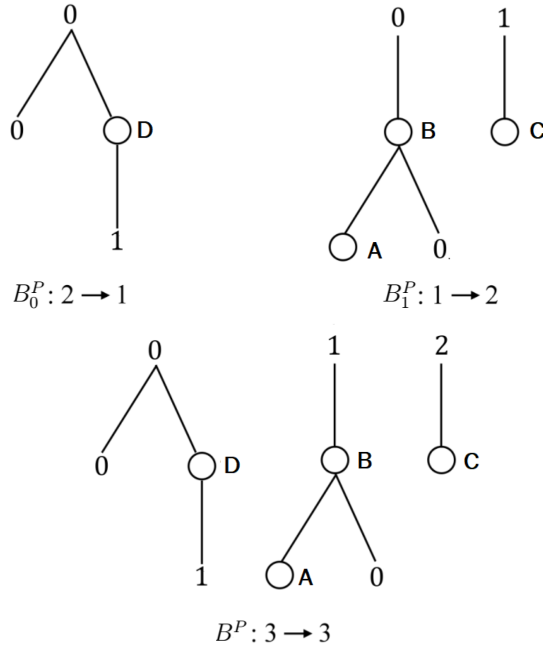


FIGURE 4.7 – Produit tensoriel de deux graphes de places : $B^P = B_0^P \otimes B_1^P$

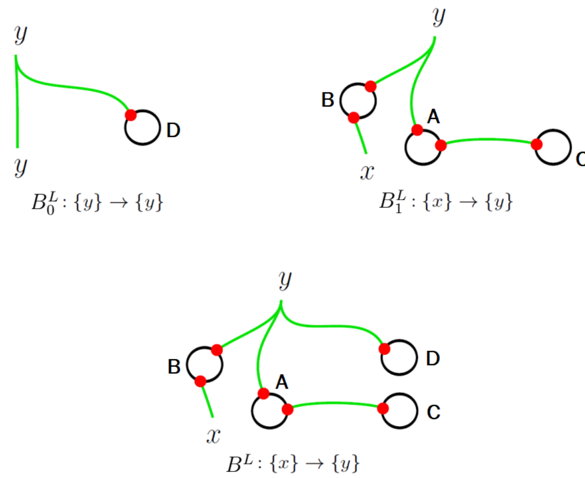


FIGURE 4.8 – Produit tensoriel de deux graphes de liens : $B^L = B_0^L \otimes B_1^L$

par :

$$B_0 \otimes B_1 \overset{\text{def}}{=} ({B_0}^P \otimes {B_1}^P, {B_0}^L \otimes {B_1}^L) :< m_0 + m_1, X_0 \uplus X_1 > \to < n_0 + n_1, Y_0 \uplus Y_1 >$$

L'exemple suivant (Figure 4.9) montre le bigraphe $B = B_0 \otimes B_1 :< 3, x > \to < 3, y >$ résultant de la composition de deux bigraphes $B_0 :< 2, y > \to < 1, y >$ et $B_1 :< 1, x > \to < 2, y >$ dont les produits tensoriels des graphes de places et des graphes de liens sont donnés dans les Figures 4.7 et 4.8 respectivement.

4.2.4 Forme algébrique

Les bigraphes disposent d'un langage de termes algébriques définissant d'autres opérations que la composition et le produit tensoriel pour les représenter structurellement. Ce langage

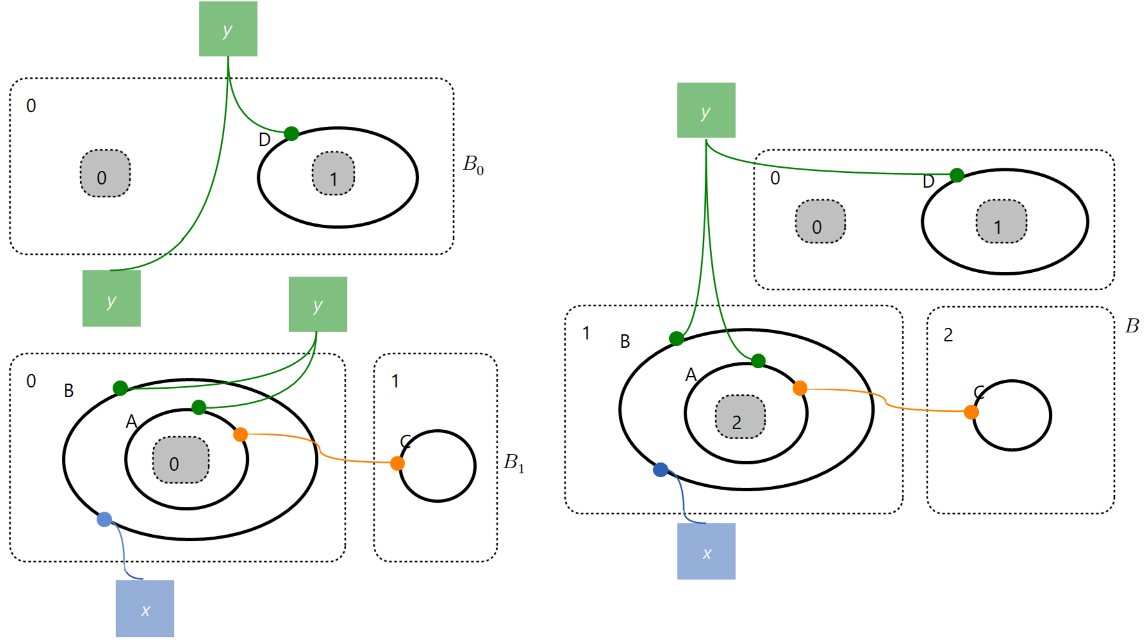


FIGURE 4.9 – Produit tensoriel de deux bigraphes : $B = B_0 \otimes B_1$

similaire à l'algèbre de processus, introduit plusieurs opérations permettant de construire des bigraphes. Dans cette section, nous donnons la définition formelle des principales opérations données par ce langage qui ne seront pertinentes pour la suite du manuscrit : le produit parallèle (noté par $\|$), la fusion (notée par $|$) et l'imbrication (notée par $\cdot$).

Définition 13 (produit parallèle) Produit parallèle des graphes de places : Si $B_i^P = (V_i, ctrl_i, prnt_i) : m_i \parallel n_i (i = 0, 1)$ sont deux graphes de places disjoints, leur produit parallèle commutatif $B_0^P \parallel B_1^P : m_o + m_1 \to n_0 + n_1$ est donné par :

$$B_0^P \parallel B_1^P \stackrel{\text{def}}{=} B_0^P \otimes B_1^P$$

Produit parallèle des graphes de liens : Si $B_i^L = (V_i, E_i, ctrl_i, prnt_i) : X_i \to Y_i (i = 0, 1)$ sont deux graphes de liens disjoints et $link_0 \cup link_1$ est une fonction de transformation sur les liens, leur produit parallèle commutatif $B_0^L \parallel B_1^L : X_o \cup X_1 \to Y_0 \cup Y_1$ est donné par :

$$B_0^L \parallel B_1^L \stackrel{\text{def}}{=} (V_0 \uplus V_1, E_0 \uplus E_1, ctrl_0 \uplus ctrl_1, link_0 \cup link_1)$$

Produit parallèle des bigraphes : Si $B_i :< m_i, X_i > \to < n_i, Y_i >, (i = 0, 1)$ sont deux bigraphes avec des supports disjoints, leur produit parallèle commutatif est donné par :

$$B_0 \parallel B_1 \stackrel{\text{def}}{=} ({B_0}^P \parallel {B_1}^P, {B_0}^L \parallel {B_1}^L) :< m_0 + m_1, X_0 \cup X_1 > \to < n_0 + n_1, Y_0 \cup Y_1 >$$

Définition 14 (fusion des bigraphes) Étant donné deux bigraphes $B_i :< m_i, X_i > \to < n_i, Y_i >$ avec $(i = 0, 1)$, supposons que $B_0 \parallel B_1$ est défini. La fusion commutative de ces deux bigraphes est donnée par :

$$B_0 \mid B_1 \stackrel{\text{def}}{=} (merge_{n_0+n_1} \otimes id_{Y_0 \cup Y_1}) \circ (B_0 \parallel B_1)$$

Avec $B_0 \mid B_1 :< m_0 + m_1, X_0 \cup X_1 > \to < 1, Y_0 \cup Y_1 >$.

Définition 15 (imbrication des bigraphes) Étant donné deux bigraphes $F : I \rightarrow < m, X >$ $etG : m \rightarrow < n, Y >$, l'imbrication de F dans G est définie par :

$$G \cdot F \stackrel{\text{def}}{=} (G \parallel id_x) \circ F : I \rightarrow < n, X \cup Y >$$

Le Tableau 4.1 [Milner, 2009] récapitule les opérations de base définies par le langage algébrique des bigraphes.

TABLE 4.1 – Principaux termes du langage algébrique des bigraphes

Terme	Forme algébrique	Forme graphique
Produit parallèle	$A_x y \, \| B_y z$	
Fusion	$A_x y \, \| B_y z$	
Imbrication	$A_x y . B_y z$	
Identité(bigraphe élémentaire)	id_i	
Site numéroté i	d_i	

4.2.5 Logique de typage (*sorting*)

En vue de fournir une description correcte d'un système donné, il est parfois nécessaire de limiter les possibilités de construction de l'ensemble de bigraphes admissibles modélisant ce système. Ceci est possible en imposant des contraintes en termes de classification des contrôles de ces bigraphes [Milner, 2008]. À ces fins, nous introduisons le concept de typage et les sortes et définissons formellement comment spécifier ce genre de restrictions. Voici quelques notations de base nécessaires : les sortes sont notées en utilisant des lettres minuscules : a, b, c. Nous écrivons $\widehat{ab}$ pour les sortes disjonctives, signifiant qu'un nœud peut être soit de sorte a ou b. Un bigraphe qui satisfait un typage Σ est dit Σ-typé.

Définition 16 (typage des places.) Un typage de places est donné par un triplé $\Sigma_P = (\Theta_P, \mathcal{K}, \Phi_P)$ où Θ_P est un ensemble non-vide de sortes et $\mathcal{K}$ est une signature Σ_P-typée qui associe une sorte à chaque contrôle et est un ensemble non-vide Φ_P de règles de formation pour Σ_P. Φ_P est une propriété d'un ensemble de bigraphes Σ_P-typés qui est satisfaite par les identités, les symétries, et préservée par la composition, le produit tensoriel et le produit parallèle. Lors de l'application d'un typage de places Σ_P sur une interface n, nous écrivons $\overrightarrow{\theta}$, où $\overrightarrow{\theta} = \theta_n \ldots \Theta_0$ énumère les sortes θ_i affectées à chaque $i \in n$.

Définition 17 (typage des liens.) Un typage de liens est donné par un triplé $\Sigma_L = (\Theta_L, \mathcal{K}, \Phi_L)$ où Θ_L est un ensemble non-vide de sortes et $\mathcal{K}$ est une signature Σ_L-typée qui associe une sorte à chaque contrôle et est un ensemble non-vide Φ_L de règles de formation pour Σ_L. Φ_L est une propriété d'un ensemble de bigraphes Σ_L-typés qui est satisfaite par les identités, les symétries, et préservée par la composition, le produit tensoriel et le produit parallèle. Lors de l'application d'un typage de liens Σ_L sur une interface $\overrightarrow{x}$, nous écrivons, $\{x_1 : \theta_n, \ldots, x_n : \theta_n\}$ où chaque $\theta_i \in \Theta_L$.

Exemple. Pour illustrer le concept de *sorting*, considérons l'exemple d'un bigraphe U modélisant le bâtiment d'une université (voir figure 4.10). Dans cette université, l'aile Ouest est réservée aux travaux pratiques sur machines, l'aile Est aux laboratoires de chimie, et où le hall central contient des amphithéâtres et des salles de cours. Le nœud de contrôle B (ou B-nœud) représentera le bâtiment en question. Les O-nœud, E-nœud et C-nœud représentent les ailes Ouest, Est ainsi que hall central du bâtiment respectivement. Un M-nœud (graphiquement représenté par un carré) représente une salle machine, un L-nœud un laboratoire (un triangle), un P-nœud un amphithéâtre (une forme ovale horizontale) et un S-nœud une salle de cours (une forme ovale verticale). Dans la bâtiment B, il y a trois entrées principales représentées par des noms internes $x_0, x_1 et x_2$ et trois sorties de secours représentées par des noms externes $y_0, y_1 et y_2$. Les noms internes et externes permettent d'accéder et de quitter l'aile Ouest, le hall central et à l'aile Est respectivement. Notons qu'en termes de places, il n'est pas admissible qu'une salle de machines se trouve dans l'aile Est ou qu'un amphithéâtre contienne un laboratoire, par exemple. Du point de vue des liens, il n'est pas accepté qu'une partie du bâtiment soit reliée à une entrée ou une sortie autre que la sienne. De ce fait, une logique de typage et de sortes s'impose pour restreindre les possibilités de modélisation, assurant ainsi une conception correcte du bigraphe U. Pour le typage de places, ceci est accompli comme suit : $\Theta_P = \{b, c, e, o, m, l, p, s\}$, $K = \{B : b, C : c, E : e, O : o, M : m, L : l, P : p, S : s\}$ et $\Phi_P = \{$tout nœud de $\widehat{mlps}$-sorte est atomique, tout fils d'un nœud de b-sorte est de $\widehat{eco}$-sorte, tout fils d'un nœud de e-sorte est de l-sorte, tout fils d'un nœud de c-sorte est de $\widehat{ps}$-sorte, tout fils d'un nœud de o-sorte est de m-sorte$\}$. Le typage de liens est construit

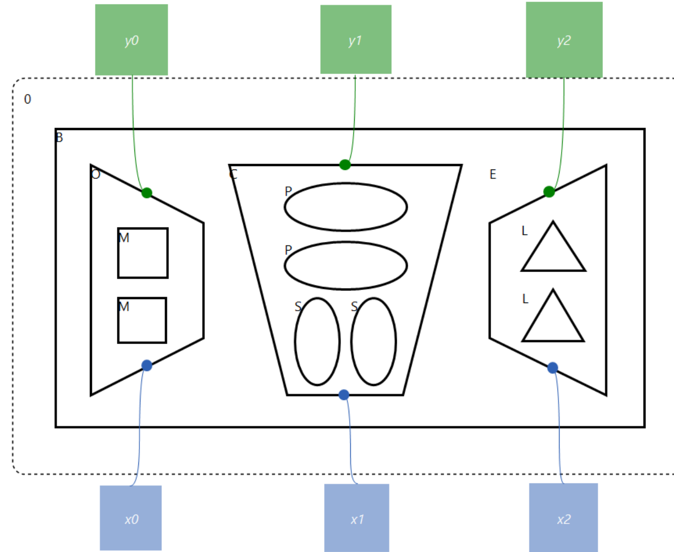


FIGURE 4.10 – Bigraphe U modélisant le bâtiment B d'une université

de la même manière en imposant des restrictions sur les liens. Par exemple : un nœud de c-sorte est relié au nom interne x_1 et au nom externe y_1.

4.2.6 Dynamique des bigraphes

Après avoir présenté les aspects structurels statiques des bigraphes, nous nous intéressons à présent à leur sémantique dynamique qui définit comment les bigraphes peuvent se reconfigurer en termes de places et de liens. Cette dynamique prend la forme de règles de réaction semblables aux règles de réécriture de graphes [Milner, 2008].

Une règle de réaction $R \to R'$ est décrite par une paire de bigraphes <redex, reactum>. Le redex est une précondition pour la réaction qui décrit un cliché qui va être changé (ou réécrit). Le reactum représente la postcondition qui spécifie le cliché après le chargement (réécriture) effectué par la réaction. On distingue deux types de reconfigurations : (1) reconfiguration sur les places par l'ajout, la suppression ou le déplacement d'un nœud (2) reconfiguration sur les liens par la création ou la destruction d'un lien entre deux nœuds ou entre un nœud et un nom interne/externe. Nous notons qu'une règle de réaction est de déclenchement spontané [Sevegnani and Calder, 2015], c'est-à-dire quand le redex est reconnu (ou matche) dans le contexte du bigraphe concerné, la réaction est directement appliquée réécrivant le redex en le reactum [Birkedal et al., 2007]. Par exemple, la règle de réaction présentée dans la figure 4.11 permet à un utilisateur (U), dans la même région qu'une salle (S), d'entrer dans la salle et se connecter à l'ordinateur (PC). C'est- à-dire, avant l'exécution de la règle, l'utilisateur est en dehors de la salle et n'est pas connecté à l'ordinateur ; après l'exécution de la règle, l'utilisateur se déplace à l'intérieur de la salle (S) et est connecté à l'ordinateur (PC) via un lien.

Définition 18 (règle de réaction.) Une règle de réaction prend la forme : $R = (R : m \to J, R' : m \to J, \eta)$ et généralement écrite $R \to R'$, où $R : m \to J$ est le bigraphe redex et $R' : m \to J$ est le bigraphe reactum. L'instanciation $\eta : m' \to m$ est une transformation d'ordinaux.

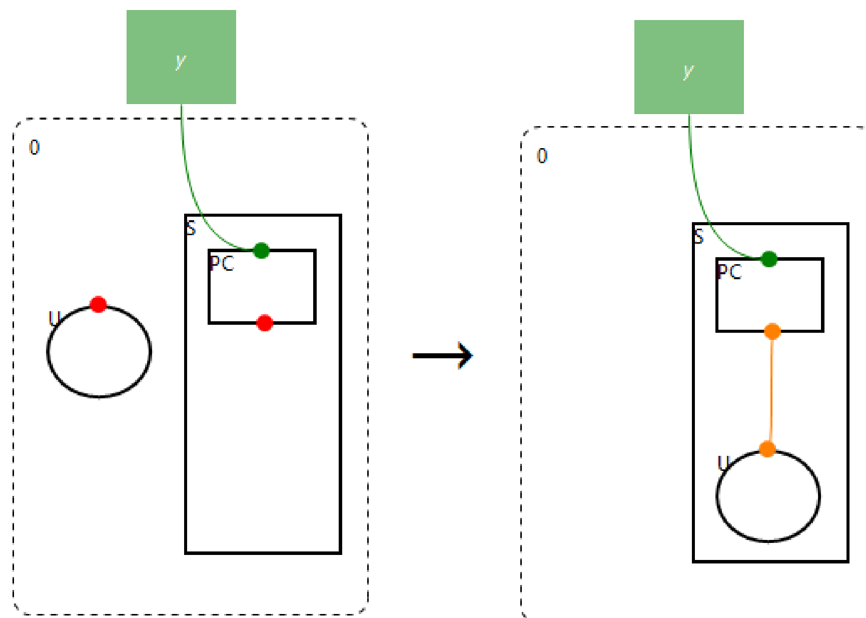


FIGURE 4.11 – Exemple d'une règle de réaction bigraphique

Après avoir abordé toutes les notions clés qui servirons de base pour la compréhension des contenus de ce manuscrit, nous définissons maintenant un système réactif bigraphique (BRS).

Définition 19 (système réactif bigraphique) . Un système réactif bigraphique (BRS) consiste en une paire (B, R), où B est un ensemble de bigraphes et R est un ensemble de règles de réaction définies sur B. La relation de réaction d'un BRS est représentée par $\rightarrow$. Nous écrivons $(B(\Sigma), R)$ lorsque les éléments de B sont Σ-typés.

4.2.7 Bigraphes concrets et bigraphes abstraits

Les bigraphes peuvent être exprimés graphiquement et algébriquement de manière abstraite ou concrète. La différence entre ces deux notions repose sur une simple subtilité [Birkedal et al., 2007] : les bigraphes concrets sont représentés en dotant les nœuds d'identifiants uniques afin de les distinguer, et en nommant les arcs internes (connexions reliant les nœuds entre eux uniquement). Les bigraphes dits abstraits représentent les nœuds par leur contrôle uniquement en ignorant leurs identifiants ainsi que ceux des arcs internes les reliant. De ce fait, ces deux représentations ont des utilités distinctes : les bigraphes concrets décrivent un cliché exhaustif d'un système quelconque tandis que les bigraphes abstraits, qui sont considérés comme une classe d'équivalence des bigraphes concrets, décrivent un système de manière plus générique.

Étant donné la nature spontanée du déclenchement des règles de réaction, écrire ces dernières de manière concrète reviendrait à spécifier un comportement spécifique pour un système précis dont les reconfigurations seraient connues et contrôlées. En revanche, spécifier les règles de réaction de manière abstraite constituerait une manière de décrire un comportement dynamique non-déterministe et concurrent, ce point est abordé de manière plus précise dans le Chapitre 6. Dans ce manuscrit, nous privilégierons la notation

abstraite des bigraphes afin d'apporter une modélisation la plus générique possible de nos systèmes.

4.2.8 Outils pratiques autour des BRS

Un nombre d'outils et environnements logiciels ont été conçus dans le but de manipuler les systèmes réactifs bigraphiques da manière concrète. Nous nous intéressons ici aux outils que nous considérons les plus pertinents tels que : BPL Tool [Højsgaard and Glenstrup, 2011], Big Red [Faithfull et al., 2013], BigMC [Perrone et al., 2012] et BigraphER [Sevegnani and Calder, 2016].

BPL Tool. Est un prototype représentant l'une des premières implémentations de systèmes réactifs bigraphiques. BPL Tool pour *Bigraphical Programming Language Tool* permet de manipuler, de simuler et de visualiser des systèmes réactifs bigraphiques. Cet outil consiste en un parseur, un moteur de *matching* et un moteur de normalisation. Il comporte une interface web et une interface en ligne de commandes. Le langage utilisé dans l'outil BPL Tool est appelé BPL. Il consiste en un ensemble de blocs de langage Standard ML (SML) [Milner et al., 1997] qui permettent d'écrire BPL directement dans SML. L'outil BPL Tool a été utilisé pour la spécification de plusieurs types de systèmes tels que : les systèmes de téléphonie mobile [Højsgaard and Glenstrup, 2011], WS-BPEL, HomeBPEL [Bundgaard et al., 2008] et les modèles platographiques [Elsborg, 2009].

BigMC. Bigraphical Model Checker [Perrone et al., 2012] est l'unique model-checker conçu pour opérer sur les systèmes réactifs bigraphiques. Il permet d'effectuer des vérifications formelles sur différents systèmes modélisés à l'aide de BRS. L'outil BigMC est basé sur le moteur de *matching* de BPL, ce qui lui permet d'explorer tous les états possibles d'un système bigraphique et de vérifier si une propriété donnée est vérifiée ou violée à chaque état du système. BigMC offre la possibilité de fournir des contre exemples ou des traces d'exécution dans le cas où la propriété n'est pas vérifiée. Les contre-exemples peuvent être utilisés pour découvrir les causalités ayant mené à des violations de la propriété. BigMC implémente une vérification explicite d'états avec un support limité pour la vérification symbolique.

Big Red. Est un éditeur graphique développé sous Eclipse qui permet de visuellement développer des bigraphes et des systèmes réactifs bigraphiques d'une manière efficace [Faithfull et al., 2013]. Notons d'ailleurs que les illustrations bigraphiques dans ce Chapitre ont été réalisées à l'aide de cet outil. Big Red est implémenté comme un ensemble de plugins Eclipse extensibles, offrant la possibilité d'être étendu avec de nouvelles propriétés et contraintes afin d'implémenter de nouvelles versions de bigraphes. Il consiste en deux plugins : un modèle bigraphique défini non modifiable indépendant des la plateforme Eclipse et un *wrapper* qui est un plugin qui implémente l'extensibilité de l'outil. De ce fait, il offre la possibilité d'intégrer le model-checker BigMC de manière à permettre son utilisation sur les modèles définis et de visualiser les résultats des analyses menées.

BigraphER. Cet outil consiste en une bibliothèque OCaml et une ligne de commandes qui permettent la manipulation, la visualisation et la simulation des systèmes réactifs bigraphiques, BRS stochastiques et les BRS avec partage [Sevegnani and Calder, 2015, Sevegnani and Calder, 2016]. BigraphER est composé de trois modules distincts : un compilateur, un moteur de matching et un moteur de réécriture. L'outil prend en entrée

un fichier source contenant la spécification du modèle. Il met en œuvre un langage de spécification similaire à la forme algébrique des bigraphes permettant de définir la signature du modèle, un ensemble de bigraphes et un ensemble de règles de réaction. Le compilateur traduit le fichier source en entrée en une représentation du modèle au moment de l'exécution. Il peut également générer une représentation graphique de chaque bigraphe spécifié dans le fichier d'entrée à l'aide du générateur automatique de graphes Graphviz [Ellson et al., 2001]. Le moteur de *matching* implémente un algorithme de reconnaissance des graphes (*matching*) basé sur un codage SAT. Il est utilisé par le moteur de réécriture pour appliquer des règles de réaction à un état et pour vérifier l'égalité des états.

4.3 Langage Maude

Maude [Clavel et al., 2007, Clavel et al., 2016, Clavel et al., 2002, Clavel et al., 1996] est un langage de haut niveau, basé sur les théories mathématiques de la logique de réécriture et la logique équationnelle. Maude est destiné à la spécification formelle des systèmes informatiques, et particulièrement les systèmes concurrents et répartis. Dans Maude, les aspects structurels et statiques d'un système sont décrits par des équations et des types de données nativement intégrés (entiers relatifs, nombres flottants, etc.). Quant aux aspects comportementaux et dynamiques, ils sont décrits par des règles de réécriture. Les spécifications réalisées dans Maude sont exécutables et peuvent êtres simulées et analysées formellement. La vérification formelle dans Maude est réalisée via un interpréteur et plusieurs techniques et outils de vérification et d'analyse formelle, tels que le model-checker basé sur la logique temporelle linéaire (LTL) et la technique de vérification par invariants. Dans cette Section, nous introduisons l'environnement Maude, ses caractéristiques et son fonctionnement. Le langage Maude est caractérisé par [Clavel et al., 2016] :

L'expressivité. Le langage Maude permet de représenter intuitivement les systèmes déterministes et non déterministes grâce aux modules fonctionnels et les modules systèmes. Le calcul déter-ministe est implémenté à l'aide d'équations dans des modules fonctionnels. Cependant, le calcul non-déterministe est représenté avec des règles de réécriture dans des modules systèmes.

La simplicité. La spécification de base de Maude est simple et facilement compréhensible. Les expressions du langage Maude (équations et règles de réécriture) ont interprétation simple : cela consiste à réécrire le côté gauche de l'expression en son côté droit.

La performance. Cette caractéristique connait une amélioration constante via l'implémentation du système Maude qui est régulièrement mise à jour. Le système Maude est en mesure d'exécuter un grand nombre de réécritures en une seconde. Ceci permet de simuler et d'analyser les différents chemins d'exécution possible dans une spécification d'une manière assez efficace.

4.3.1 Syntaxe et notations

Dans le langage Maude, les unités basiques de spécification ou de programmation sont appelées *modules*. On distingue deux principaux types de modules : les modules

fonctionnels, qui implémentent des théories équationnelles et les modules systèmes, qui implémentent des théories de réécriture définissant le comportement dynamique d'un système. Dans cette Section, nous fournissons les notations et les aspects syntaxiques de base des différents modules Maude.

4.3.2 Modules fonctionnels

Ces modules définissent les types de données et les opérations qui sont utilisées par les équations (conditionnelles et non conditionnelles). À savoir : les sortes, les sous sortes et les opérations. Les attributs des opérations sont définis sous Maude à l'aide des mots clés : ctor (constructeur), assoc (associative), comm (commutative), id (élément neutre) et prec (degré de précédence ou de priorité). Un module fonctionnel dans Maude est déclaré avec le mot-clé fmod, suivi de nom du module :

```
fmod <nom-module> is <déclarations et expressions> endfm .
```

Les déclarations et les expressions dans un module fonctionnel peuvent être :

Des importations d'autres modules. Des modules fonctionnels prédéfinis peuvent être importés en utilisant les primitives protecting, including ou extending. Des déclarations de sortes et sous-sortes. Les sortes (les types de données) sont déclarées avec le mot-clé sort, et les sous-sortes sont définis en utilisant subsort. Dans le cas où nous aurions plusieurs sortes ou sous-sortes à déclarer, nous utilisons le mot-clé sorts ou subsorts.

Des déclarations d'opérations. Les opérations qui agissent sur les sortes et les sous-sortes sont déclarées à l'aide du mot-clé op.

Des déclarations d'équations. Les équations peuvent être conditionnelles ou non. Une équation non conditionnelle suit la syntaxe suivante :

```
eq <Terme-1>=<Terme-2>[<Attributs>] .
```

Les deux termes Terme-1 et Terme-2 dans l'équation Terme-1 = Terme-2 doivent être de même sorte. Pour qu'une équation soit exécutable, toutes les variables dans Terme-2 (la partie droite de l'équation) doivent apparaître dans Terme-1 (la partie gauche). Les équations conditionnelles prennent la forme suivante :

```
ceq <Terme-1>=<Terme-2> if <EqCondition-1>...<EqCondition-k> <Attributs>].
```

La syntaxe des conditions d'une équation conditionnelle est de trois variantes :

— Équations ordinaires de la forme t = t',
— Équations de correspondance (matching) t : = t'
— Équations booléennes de la forme t, où t est un terme algébrique de genre [Bool], équivalent à `t = true`.

4.3.3 Modules systèmes

Un module système dans Maude implémente une théorie de réécriture. Dans une théorie de réécriture on retrouve des sortes, des types, des opérateurs, des déclarations d'équations, et des règles de réécriture, qui peuvent être conditionnelles ou non. Par conséquent, toute théorie de réécriture a une théorie équationnelle sous-jacente. Les transitions concurrentes locales dans un module système sont représentées par des règles de réécriture. Les règles peuvent être exécutées si la partie gauche d'une règle correspond à un fragment de l'état du système, et si la condition de la règle est satisfaite. La transition spécifiée par la règle pourrait être appliquée, et le fragment identifié de l'état se transforme en l'instance correspondante du côté droit. La réécriture dans ces modules est basée sur la simplification équationnelle : pour atteindre la forme réduite et finale, dite canonique d'une expression, l'application des règles de réécriture est répétée jusqu'à ce que aucune simplification ne soit applicable. Ainsi, chaque classe d'équivalence a un représentant canonique unique qui peut être calculé par la simplification équationnelle dynamisée par les règles de réécriture. Un module système Maude est déclaré selon la syntaxe :

```
mod <nom-module> is <déclarations-et-expressions> endm .
```

Un module système dans Maude est déclaré avec le mot-clé mod, suivi de nom du module. Les déclarations et les expressions dans un module fonctionnel peuvent être : des importations d'autres modules fonctionnels ou systèmes, des déclarations des sortes ou des sous-sortes, déclarations des opérations, déclarations des équations et déclarations des règles de réécriture conditionnelles. Les règles de réécriture conditionnelles et non conditionnelles déclarées au sein d'un module système prennent la syntaxe suivante :

```
rl [<Étiquette>] : <Terme-1> => <Terme-2> [<Attributs>] .
crl [<Étiquette>] : <Terme-1> => <Terme-2> if <Condition-1>
                ... <Condition-k>[<Attributs>] .
```

Les deux termes `<Terme-1>` et `<Terme-2>` sont des termes de même sorte qui peuvent contenir des variables. `<Étiquette>` est l'étiquette de la règle de réécriture ; elle peut être omise. Les conditions d'une règle de réécriture conditionnelle `<Condition-k>` peuvent contenir des expressions de réécriture qui testent la possibilité de réécrire des termes algébriques.

Exemple (Utilisateur se connectant à un ordinateur). Reprenons l'exemple simple de l'utilisateur qui se connecte à un ordinateur se trouvant dans une salle (voir Section 4.2). On voudrait étendre le précédent exemple de manière à ne permettre à l'utilisateur de se connecter que s'il a l'autorisation de le faire et si l'ordinateur en question est libre. Supposons que l'utilisateur dispose d'un quota horaire définissant son droit de connexion à l'ordinateur (càd. Quand le quota > 0). Toutefois, il ne peut se connecter que si l'ordinateur en question est libre.

Dans le module fonctionnel (voir listing 4.1), la salle est représentée par la sorte S, l'utilisateur par la sorte U et l'ordinateur par la sorte PC. Les opérations se terminant par `[ctor]` introduisent les constructeurs de chaque sorte, c'est-à-dire, une définition de l'axiome permettant de reconnaitre un terme exprimant une sorte correspondante. Ainsi, une salle pouvant contenir un ordinateur et un utilisateur est exprimée par : `S< _ | _ > : U PC -> S`. Une opération quelconque est définie en deux temps : premièrement, il faut spécifier sa signature (nom, types des sortes en entrée et la sorte en sortie)

ensuite, on la définit avec des équations de manière à définir sa forme générale et ses cas particuliers. L'option [owise] pour *otherwise* ou « autrement » est utilisée pour donner à une équation une valeur par défaut, dans le cas où une possibilité d'évaluation n'aurait pas été spécifiée. Pour l'opération *canConnect(user,PC)* exprime la condition « l'utilisateur peut se connecter si son quota est positif et si l'ordinateur est libre ». Elle est calculée à partir de la conjonction des conditions ($quota > 0$) et (*not isUsed(PC)*).

```
1 fmod connect_Fun is
2 protecting BOOL . protecting FLOAT .
3 sorts S PC U .
4 ---constructeurs
5 op salle <_|_> : U PC -> S [ctor] .
6 op PC<_> : U -> PC [ctor] .
7 op U<_> : Float -> U [ctor] .
8 op noUser : -> U [ctor] .
9 op noPC : -> PC [ctor] .
10 ---signature des equations
11 op isUsed(_) : PC -> Bool .
12 op canConnect(_,_) : U PC -> Bool .
13 ---declaration de variables
14 var quota : Float .
15 var user : U .
16 var pc : PC .
17 ---implementation des equations
18 eq isUsed(noPC) = false .
19 ceq isUsed(PC< user >) = false if user == noUser .
20 eq isUsed(PC< user >) = true [owise] .
21 eq canConnect(noUser,noPC) = false .
22 eq canConnect(noUser,pc) = false .
23 eq canConnect(user,noPC) = false .
24 ceq canConnect(user,pc) = false if isUsed(pc) .
25 ceq canConnect(U< quota >,pc) = true if (quota > 0.0 and not isUsed(pc)) .
26 eq canConnect(user,pc) = false [owise] .
27 endfm
```

Listing 4.1 – Module fonctionnel connect_Fun

```
1 mod connect_Sys is
2 including cnnect_Fun .
3 ---declaration des variables
4 vars user user2 : U .
5 ---implementation des regles de reecriture
6 crl [connect] : salle< user | PC< user2 > >
7 => salle< noUser | PC< user > >
8 if canConnect( user , PC< user2 > ) .
9 crl [disconnect] : salle< noUser | PC< user > >
10 => salle< user | PC< noUser > >
11 if (not canConnect( user , PC< user > )) .
12 endm
```

Listing 4.2 – Module système connect_ Sys

Il devient maintenant possible de définir des règles de réécriture à déclenchement conditionnel. Dans le module système (voir listing 4.2), nous définissons une règle pour

contrôler la connexion d'un utilisateur à un ordinateur et une règle pour forcer un utilisateur à se déconnecter quand son quota est épuisé.

4.3.4 Vérification formelle dans Maude

Un module système définit une théorie de réécriture décrivant un modèle mathématique exécutable. De ce fait, Maude intègre plusieurs types de vérification formelle en vue d'analyser le comportement des systèmes modélisés. Ainsi, des modules de vérification formelle peuvent être spécifiés dans Maude afin de définir certaines propriétés liées au comportement d'un système. Ceci permet de vérifier si ces propriétés sont assurées durant l'exécution et, dans le cas contraire, de fournir des contre-exemples montrant leur violation.

Le système Maude fournit de nombreux outils et implémente un certain nombre de techniques d'analyse formelle des comportements de systèmes, tels que la vérification par invariants, le prouveur de théorèmes, le model-checker LTL, l'analyse de terminaison et l'analyse de cohérence. Nous détaillons dans ce qui suit le model-checker LTL, en raison de sa pertinence avec le travail présenté dans cette thèse. Pour plus de détails sur les autres techniques, nous invitons le lecteur à consulter [Clavel et al., 2007, Clavel et al., 2016].

Le model-checker Maude est basé sur la logique temporelle linéaire (LTL). Cette technique de vérification formelle est riche, intuitive et largement utilisée pour la spécification et la vérification de différents types de propriétés. Par exemple, les propriétés de sûreté (quelque chose de mauvais n'arrive jamais), de vivacité (quelque chose de bon finit par arriver) et d'équité (quelque chose de bon se répète infiniment). Pour exploiter LTL, Maude fournit la possibilité de décrire une structure de Kripke [Baier and Katoen, 2008]. Une structure de Kripke définit un modèle de logique temporelle de la manière suivante :

Définition 20 (Structure de Kripke) . Étant donné un ensemble AP de propositions élémentaires, une structure de Kripke est définie par $A = (A, \rightarrow_A, L)$. Où A est l'ensemble des états du système, $\rightarrow_A$ est une relation de transition, et $L : A \rightarrow AP$ est une fonction d'étiquetage associant à chaque état $a \in A$, un ensemble $L(a)$ de propositions élémentaires dans AP qui sont satisfaites à l'état a. $LTL(AP)$ définit les formules de la logique temporelle linéaire propositionnelle. La sémantique de $LTL(AP)$ est définie par la relation de satisfaction : $A, a \vDash \varphi$, où $\varphi \in LTL(AP)$.

Une structure de Kripke permet de décrire le comportement d'un système à l'aide d'un système de transition [Schoren, 2011] où les nœuds représentent les états du système et où les arcs représentent les règles de réécritures définies dans le module système. Dans Maude, une structure de Kripke correspond à un module fonctionnel utilisé pour définir les propriétés souhaitées ainsi que les formules LTL associées. Ainsi, les formules de la logique temporelle linéaire propositionnelle LTL(AP) sont nativement définies dans Maude comme suit :

- True : $T \in LTL(AP)$.
- Propositions élémentaires : si $p \in AP$, alors $p \in LTL(AP)$.
- Opérateur Next : si $\varphi \in LTL(AP)$ alors $\bigcirc\varphi \in LTL(AP)$.
- Opérateur Until : si $\varphi, \psi \in LTL(AP)$, alors $\varphi \cup \psi \in LTL(AP)$.
- Opérateurs logiques : si $\varphi, \psi \in LTL(AP)$, alors les formules : $\neg\varphi$, et $varphi \vee \psi$ appartiennent à LTL(AP).

D'autres opérateurs et conjonctions LTL peuvent être définis à l'aide de l'ensemble des conjonctions présentées comme suit :

Opérateurs booléens :

- False = $\neg T$.
- Conjunction : $\varphi \wedge \psi = \neg((\neg\varphi) \vee (\neg\psi))$
- Implication : $\varphi \rightarrow \psi = \neg(\neg\varphi) \vee \psi$.

Quelques opérateurs temporels supplémentaires :

- Eventuellement (Eventually) : $<> \varphi = T \cup \varphi$.
- Toujours (Henceforth) : $[\,]\varphi = \neg <> \neg\varphi$.

La signature LTL de la syntaxe mathématique ci-dessus est définie dans Maude à travers un module fonctionnel déclaré dans le fichier "model-checker.maude".

De cette manière, un concepteur peut définir ses propres propositions dans AP via des équations conditionnelles, de la manière suivante :

ceq <terme> |= <φ_i> = true if <condition> == true .

Où terme est un fragment du système représentant un état à étiqueter avec la proposition φ, si la condition (définie comme une équation dans le module fonctionnelle est satisfaite). Il est également possible de définir des axiomes (propositions supposées vraies) via des équations non conditionnelles. Ensuite, les formules LTL propositionnelles sont définies par des équations de la manière suivante :

eq <formule> = [φ_1 | OP1] [φ_2] OP2 φ_3 ... OPm φ_n.

Où formule est une propriété de LTL(AP) à définir, `OP1,2...m` sont des opérateurs de la logique LTL et $\varphi_1, \varphi_2, ...\varphi_n$sont des propositions élémentaires de AP.

Le model-checker de Maude. Le model-checker LTL de Maude est exécuté avec, en paramètres : (1) un état E du système en guise d'état initial pour la vérification et (2) une formule F dans LTL(AP) à vérifier. La primitive « True » sera affichée si aucune violation de F n'est détectée durant l'exécution du système (via le déclenchement des règles de réécriture) à partir de l'état E. Dans le cas où F serait violée pendant l'exécution, le model-checker affichera un contre-exemple montrant la trace d'exécution ayant mené à cette violation. Notons que le fragment (terme) décrivant E, doit être d'une sorte qui appartient à l'union des sortes des termes ayant servis à définir les propositions atomiques.

4.4 Théorie des files d'attente

La théorie des files d'attente [Kleinrock, 1976, Berry, 2006, Baynat, 2000] est un outil mathématique servant à étudier les congestions qui peuvent se produire dans un système donné. Initialement introduite par le mathématicien Danois Erlang afin d'étudier la gestion des réseaux téléphoniques, ce n'est qu'après les apports d'autres mathématiciens dont Kendall et Kolmogorov que la théorie des files d'attente s'est vraiment développée. De nos jours, on retrouve le raisonnement introduit par cette théorie dans l'analyse de phénomènes dans plusieurs domaines : la gestion des avions au décollage ou à l'atterrissage, l'attente des clients aux guichets [Joustra and Van Dijk, 2001], l'estimation de

la satisfaction des clients [Nosek Jr and Wilson, 2001], le stockage des programmes informatiques avant leur traitement, ou encore l'ordonnancement des processus avant leur exécution. En effet, il est intéressant d'observer que des files d'attentes se forment même dans des systèmes non congestionnés. Un exemple simple [Gueroui, 2015] est celui d'un établissement de restauration rapide qui peut traiter en moyenne 200 commandes par heure. On y observe tout de même la formation de files d'attente avec un nombre moyen de 150 commandes. L'expression clé ici est « en moyenne ». Le problème est dû au fait que les arrivées des clients ont lieu à des intervalles aléatoires et variables plutôt qu'à intervalles fixes et régulières. De plus, certaines commandes nécessitent un temps de traitement plus important que d'autres. De cette réalité, on distingue alors deux notions clés dans cette théorie : le processus d'arrivée et de service qui décrivent la manière dont les clients pénètrent et quittent le système en question. Étant donné que ces deux processus possèdent un degré de variabilité élevé, le système est soit temporairement congestionné, soit vide (si aucun client ne se présente). Dans cette Section, nous introduirons cette théorie, son objectif, ses caractéristiques et discuterons son apport quant aux travaux faisant l'objet de ce manuscrit.

4.4.1 Objectif de l'analyse des files d'attente

L'objectif de l'analyse des files d'attente est de minimiser le coût total lié au fonctionnement d'un système donné [Gueroui, 2015]. Ce coût est la somme de deux coûts : le coût de service, lié à la capacité de service mise en place et le coût d'attente, lié à l'attente des clients en vue de consommer ce service.

— *Le coût de service* : résulte du maintien d'un certain niveau de service (e.g. le nombre de caisses ouvertes dans un supermarché ou de guichets ouverts dans une banque, etc.).

— *Le coût d'attente* : est constitué des salaires payés aux employés qui attendent pour effectuer leur travail (e.g. un boulanger qui attend la livraison de matières premières), du coût de l'espace disponible pour l'attente (e.g. la grandeur de la salle d'attente d'un cabinet médical) et le coût associé à la perte de clients impatients qui partent.

En pratique, les ressources inoccupées définissent une capacité perdue (car non-stockable) qui impacte considérablement le coût de service. D'un autre côté, le client d'un service est généralement externe à celui-ci, ce qui rend le coût d'attente difficilement quantifiable. De ce fait, le temps d'attente est généralement considéré comme un critère de mesure du niveau de service. Le gestionnaire décide d'un temps d'attente « tolérable » selon les situations (e.g. il n'est pas tolérable d'attendre plus d'un quart d'heure dans un établissement de restauration dit rapide) et déploie la capacité susceptible d'assurer le niveau de service adéquat. Finalement, l'objectif de l'analyse des files d'attente est de trouver un compromis, entre le coût de service et le coût d'attente, ce qui permettrait de minimiser le coût total d'un système tout en maximisant le niveau de service au possible [Phillips, 2004]. La figure 4.12 illustre bien ce concept. Le coût de service (représenté de manière linéaire par simplicité) augmente lorsque la capacité de service augmente. En revanche, lorsque la capacité de service augmente, le coût d'attente et le nombre de clients en attente tendent à diminuer. Le coût total (somme des coûts de service et d'attente) est représenté par une courbe en forme de U. Le niveau de service optimal correspond à la configuration où le coût total serait à son minimum.

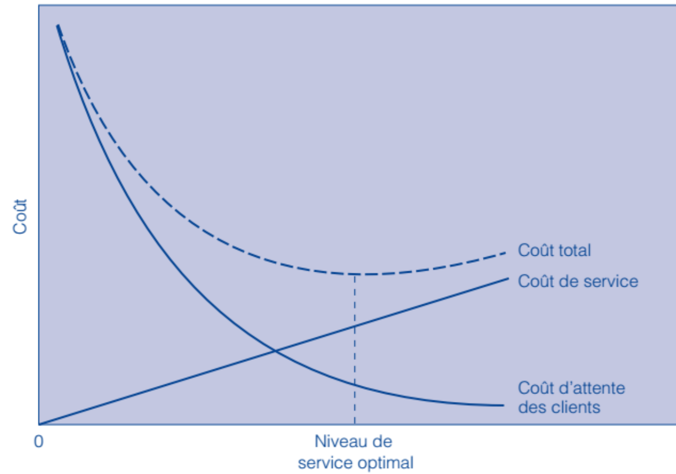


FIGURE 4.12 – Analyse des files d'attente [Gueroui, 2015]

4.4.2 Caractéristiques des systèmes de files d'attente

Analyser efficacement un système de files d'attente repose sur le choix d'un modèle d'analyse approprié. Dans le cadre de la théorie des files d'attente, plusieurs modèles d'analyse ont été conçus, prenant en compte quatre caractéristiques principales [Baynat, 2000] : (1) la population, (2) le nombre de serveurs, (3) les processus d'arrivée et de service des clients et (4) la discipline de service des clients. La figure 4.13 illustre la forme générale d'un système de files d'attente.

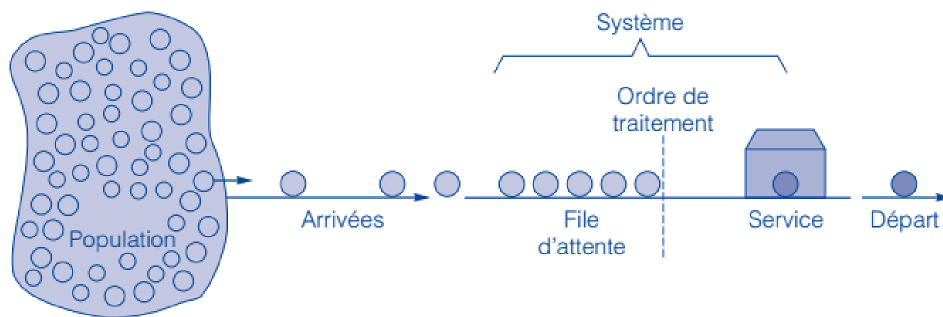


FIGURE 4.13 – Système de file d'attente simple [Gueroui, 2015]

La population. Désigne la source de clients potentiels du système, elle peut être supposément infinie ou illimitée (e.g. clients d'un supermarché) ou finie, si le nombre de clients est limité (e.g. nombre de voitures en réparation chez un garagiste).

Le nombre de serveurs. Un serveur représente une ressource ou une entité offrant un service, la capacité de service est donc définie par la capacité de chaque serveur et le nombre de serveurs disponibles. Les systèmes de files d'attente peuvent fonctionner avec un serveur unique (e.g. caisse unique dans un petit magasin) ou avec de multiples serveurs opérant de manière parallèle et généralement offrant les même services (e.g. les caisses d'un grand supermarché). La figure 4.14 illustre ces deux cas de figure.

Les processus d'arrivée et de service. Les files d'attente résultent de la variabilité des tendances (processus) d'arrivée et de service des clients dans le système. Elle se forment

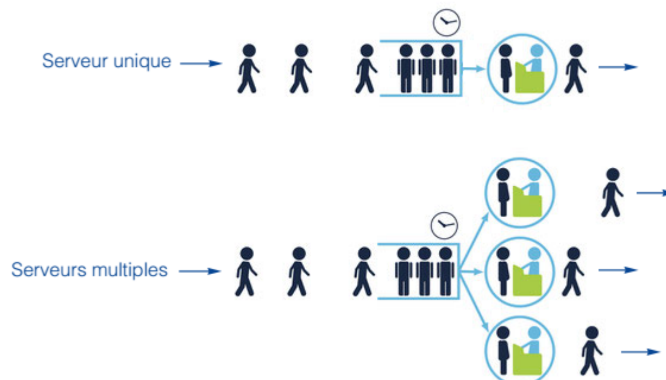


FIGURE 4.14 – Nombre de serveurs dans un système de files d'attente

du fait du degré élevé de variations dans les arrivées et dans le temps de service, causant ainsi des congestions dans le système. Les variations liées à ces deux processus peuvent être représentées par des distributions théoriques de probabilités. Dans le processus d'arrivée, le nombre d'arrivées dans un intervalle donné suit la loi de Poisson alors que le processus de service suit une loi exponentielle. Une file d'attente se forme potentiellement lorsque les arrivées arrivent en groupe ou quand le temps de service est particulièrement élevé ; elle se forme très probablement lorsque ces deux facteurs se manifestent.

La discipline de service. Cette discipline décrit l'ordre d'ordonnancement des clients dans le système. La plus commune étant : premier arrivé, premier servi (PEPS) ou FIFO pour *first-in, first-out* en anglais. Cependant, d'autres disciplines sont utilisées comme l'ordonnancement par priorité (e.g. service d'urgences d'un hôpital) ou encore, l'ordonnancement par temps de service (e.g. l'algorithme SJF pour *shortest job first* ou tâche la plus courte d'abord dans les processeurs informatiques).

4.4.3 Modèles de files d'attente

Il est possible de représenter plusieurs modèles de files d'attente en se basant sur la notation A/S/K/Q/P/D introduite par Kendall [Kendall, 1953] où :

- A est le processus d'arrivée, généralement représenté par une distribution aléatoire suivant la loi de Poisson, autour d'un taux moyen λ.
- S est le processus de service, généralement représenté par une distribution exponentielle autour d'un taux moyen μ.
- K est le nombre de serveurs (avec $K > 0$).
- Q est la taille de la file d'attente (pouvant être finie ou infinie).
- P est la taille de la population représentant les clients potentiels du système (pouvant être finie ou infinie).
- D est la discipline de service (e.g. FIFO, SJF, etc.) décrivant l'ordonnancement du traitement des clients par le système.

Certains modèles, dits classiques, sont très souvent utilisés tels que le modèle $M/M/1$ et $M/M/\infty$ représentant respectivement (en utilisant la notation de Kendall abrégée) les systèmes de files d'attente avec un serveur et avec un nombre supposément infini de

serveurs. La notation M pour les processus d'arrivée et de service est une convention indiquant que le premier suit une loi de Poisson et que le deuxième suit une loi exponentielle. Dans ces deux exemples, les tailles de la file d'attente et de la population sont supposées infinies et la discipline de service est FIFO. La théorie des files d'attente relève du domaine des probabilités. De ce fait, elle fournit un moyen de calculer de manière mathématique certaines mesures de performance telles que : (1) le nombre moyen de clients en attente dans la file ou dans le système, (2) le temps moyen d'attente en file et dans le système, (3) le taux d'utilisation du système (càd. Le pourcentage de la capacité utilisée), (4) le coût lié au niveau de service (capacité) mis en place et (5) la probabilité qu'un client potentiel attende pour être servi. De cette manière, il devient possible de modéliser un système de files d'attente en « jouant » sur les valeurs des différents paramètres le définissant. Le but étant, comme expliqué plus tôt, de trouver la ou les configurations qui permettraient d'avoir le coût total le plus avantageux pour un niveau de service optimal, pour des taux d'arrivée (λ) et de service (μ) donnés.

4.4.4 Théorie des files d'attente et gestion dynamique des ressources

Dans le monde des systèmes informatiques, il est possible d'appliquer une vision basée files d'attente de plusieurs manières. Par exemple, on peut considérer le système comme un ordinateur et les clients comme des processus. De cette façon, la fille d'attente en elle-même serait le *buffer* des processus en attente d'exécution par le processeur, et la discipline de service serait l'algorithme d'ordonnancement du *scheduleur* (ou ordonnanceur) de ce dernier. La population potentielle ici est limitée car dépendante de la taille de la mémoire RAM (limitant ainsi le nombre de programmes pouvant être lancés). La taille de la file d'attente (buffer) est également limitée.

Dans nos travaux, nous nous intéressons à une vision plus large qui est celle des systèmes distribués (voir Figure 4.15). Dans de tels systèmes, les clients sont les requêtes envoyées par les utilisateurs finaux du système, ce qui rend la population potentiellement infinie. Une fois dans le système, les requêtes forment une file d'attente au niveau du *Load Balancer* (LB) ou équilibreur de charge avant d'être réparties sur les différents serveurs mis à disposition dans le système (au niveau desquels d'autres files d'attentes peuvent très bien se former). La discipline de service est décidée par les algorithmes de *Load Balancing* ou équilibrage de charge du LB.

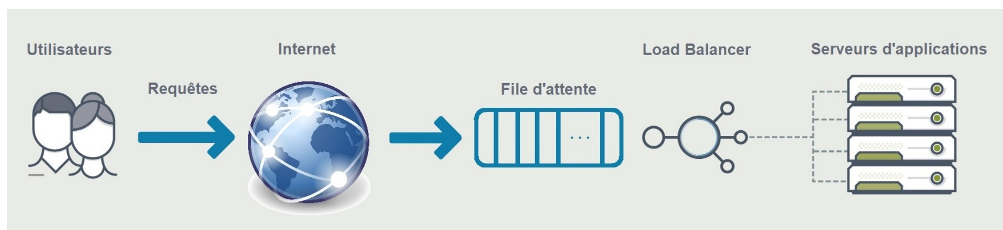


FIGURE 4.15 – Répartition des requêtes dans un système distribué

Dans cet exemple, il est intéressant d'observer les faits suivants : (1) l'arrivée des requêtes est extrêmement variable et difficile à prédire et (2) le nombre de serveurs est parfois décidé de manière dynamique ou à la demande. En effet, dans les systèmes élastiques (cf. Section 2.3) par exemple, les ressources (serveurs) sont provisionnées et libérées de manière dynamique en réponse aux fluctuations observées au niveau des arrivées des

requêtes. Il serait donc logique de penser qu'une bonne gestion des ressources implique un taux d'utilisation de 100%. Cependant, il est important de garder en tête que le fait d'augmenter le taux d'utilisation revient à augmenter à la fois le nombre de requêtes en attente et le temps moyen d'attente [Gueroui, 2015]. En réalité, ces deux mesures augmentent indéfiniment lorsque le taux d'utilisation approche de 100%. Si tous les serveurs sont occupés, il est certain que les requêtes en entrée vont attendre. Cela implique que dans le monde réel, un taux d'utilisation de 100% est irréaliste surtout en considérant la nature hautement variable des affluences des requêtes dans le système (qui peut être vide par moments). Le but d'une bonne analyse, comme expliqué précédemment, serait plutôt d'équilibrer le système de manière à minimiser le coût total.

Modèle de files d'attente avec nombre variable de serveurs. Si le taux d'utilisation à 100% reste irréaliste, il peut cependant être maximisé et les coûts de service minimisés. En effet, en plus des modèles classiques de files d'attente, il existe également des modèles dits avec nombre de serveurs à la demande comme présenté dans [Mazalov and Gurtov, 2012, Młyńczak, 2007]. Étant donné la nature non-stockable des ressources et les coûts perdus liés à leur non-utilisation, ce modèle consiste à augmenter et réduire le nombre de serveurs déployés de manière à s'adapter à la dynamique du système, caractérisée par les requêtes en entrée et requêtes traitées. Ces deux évènements sont généralement considérés comme des processus de vie et de mort (*Life and Death Process*) et sont modélisés par des chaines de Markov, notamment selon les équations de Chapman-Kolmogorov [Kleinrock, 2005]. En vue de la complexité du calcul [Kleinrock, 2005] lié aux chaînes de Markov et de son impact sur les performances liées à la décision, cette modélisation est considérée idéale par [Młyńczak, 2007] pour un nombre de serveurs dans l'intervalle [1, 2]. Au-delà, les auteurs la jugent irréaliste et estiment que le système modélisé est difficilement contrôlable et que le calcul du coût total optimal lié à son fonctionnement est une tâche difficile.

4.5 Conclusion

Dans ce Chapitre, nous avons introduit les fondements formels adoptés dans notre travail. Dans un premier temps, nous avons présenté les systèmes réactifs bigraphiques. Nous avons décrit l'anatomie des bigraphes, leur forme graphique et définitions formelles. Nous avons également exposé à travers des exemples les principales opérations qui peuvent être effectuées sur les bigraphes. Ensuite, nous avons introduit leur forme algébrique, ainsi que le principe de typage des bigraphes. Enfin, nous avons présenté l'aspect dynamique des systèmes réactifs bigraphiques et les principaux outils autour de ce formalisme. Nous avons aussi présenté les différents concepts relatifs au langage Maude. Plus précisément, nous avons introduit la syntaxe et les notations du langage Maude et les techniques d'analyses formelles dans le système Maude. Finalement, nous avons introduit la théorie des files d'attente en présentant sa philosophie, ses caractéristiques et son utilisation pour l'analyse des performances des systèmes.

Deuxième partie

Contributions

Chapitre 5

Méthodologie et contributions

Ce Chapitre introductif de la partie Contributions a pour but de donner une vision d'ensemble des travaux présentés dans ce manuscrit. Dans un premier temps, nous présentons un bref rappel du contexte où s'inscrivent nos travaux et des objectifs que nous tentons d'atteindre (Section 5.1). Ensuite, nous donnons une vue d'ensemble de notre approche pour la formalisation et l'évaluation de l'élasticité multi-couches des systèmes Cloud dans la Section 5.2. Nous y expliquons la méthodologie suivie afin de répondre aux problématiques identifiées. Enfin, la Section 5.3 détaille la répartition des contributions présentées à travers les différents Chapitres qui constituent ce manuscrit.

Sommaire

5.1 Contexte et objectifs 76
5.2 Méthodologie et principes de la solution 77
5.2.1 Modélisation formelle des systèmes Cloud élastiques 77
5.2.2 Exécution et vérification de l'élasticité 78
5.2.3 Évaluation de l'élasticité 79
5.3 Organisation des contributions 80

5.1 Contexte et objectifs

Les systèmes Cloud élastiques évoluent dans un environnement particulièrement dynamique et variable. De ce fait, ils se distinguent des autres modèles informatiques par une complexité considérable au niveau de la compréhension et du contrôle de leurs comportements. Le comportement élastique autonomique d'un système Cloud (en termes de dimensionnement des ressources) dépend de la combinaison d'une multitude de facteurs tels que la quantité de ressources disponibles, la charge de travail en entrée, la logique gouvernant le comportement du contrôleur d'élasticité ou encore les différentes politiques de haut niveau et propriétés à satisfaire. La complexité de ces dépendances, ainsi que la maitrise des potentiels effets de bords néfastes sur l'état global du système, rendent la conception des systèmes Cloud élastique très difficile. Du point de vue de la spécification, de la vérification et de l'évaluation, nous considérons quatre objectifs de recherche (OR) :

OR1- Définir les comportements élastiques multi-couches d'un système Cloud.
OR2- Assurer une exécution autonomique de ces comportements.
OR3- Vérifier le bon fonctionnement de ces comportements.
OR4- Évaluer les performances et les coûts liés à ces comportements.

Nos travaux visent à fournir une approche de modélisation formelle des systèmes Cloud et de leur élasticité en couvrant toutes les phases du processus de développement, de la spécification à l'évaluation quantitative des comportements en passant par leur vérification qualitative. Dans la Section suivante, nous donnons une vision d'ensemble de notre solutions proposée. Nous en présentons les principales phases, leurs objectifs ainsi que l'apport des modèles et théorie formels utilisés. Enfin, nous expliquons l'organisation de la suite du manuscrit en termes de répartition des contributions, à travers les Chapitres à venir.

5.2 Méthodologie et principes de la solution

En vue d'adresser les problématiques identifiées, notre solution se décompose en trois grandes phases (cf. Figure 5.1). Dans un premier temps, nous proposons une spécification formelle des sytèmes Cloud d'un point de vue structurel et comportemental. Ensuite, nous vérifions le bon fonctionnement des comportements décrits à travers leur *vérification qualitative*. Enfin, nous procédons à une *évaluation quantitative* des comportements définis afin de valider les résultats obtenus.

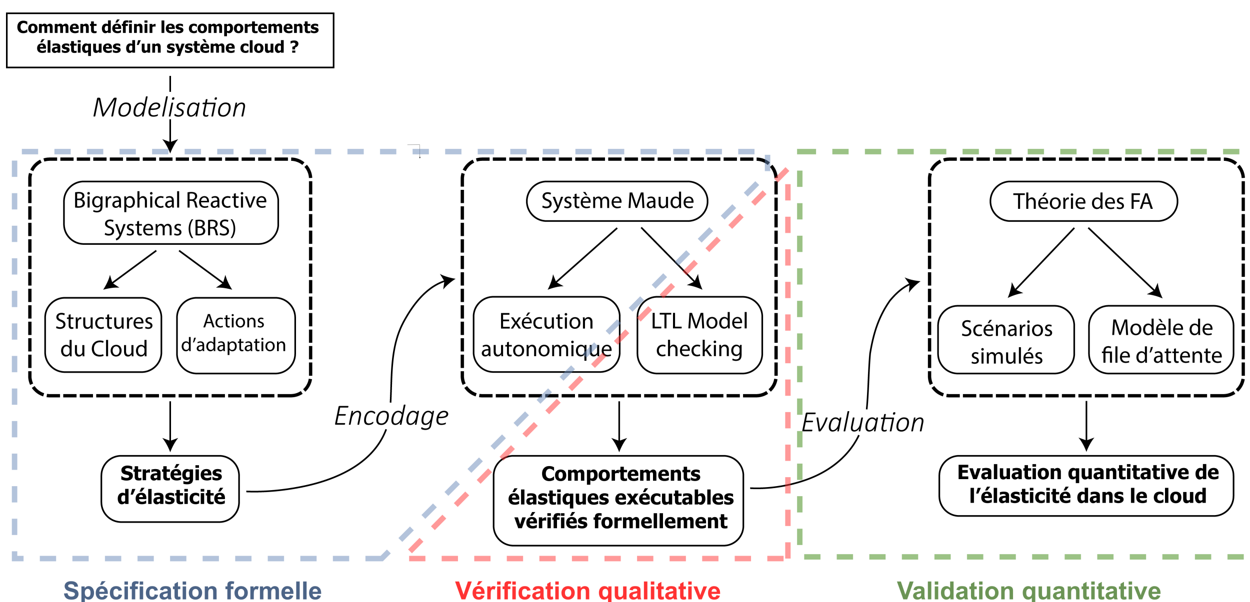


Figure 5.1 – Vue d'ensemble des contributions

5.2.1 Modélisation formelle des systèmes Cloud élastiques

Cette phase vise à fournir une description formelle des systèmes Cloud et de leurs comportements élastiques. Ceci est accompli en deux grandes étapes, qui sont la modélisation et l'encodage, de la manière suivante :

Étape de modélisation. En se basant sur le formalisme BRS, le but de cette étape est de fournir une modélisation de la structure des systèmes Cloud d'un côté et de leurs

comportements d'un autre. Précisément, nous utilisons les bigraphes pour apporter une description formelle des systèmes Cloud computing, en se focalisant sur la localité et la connectivité des entités les composant aux niveaux infrastructure (i.e. VMs) et application (i.e. instances de services). Ceci se traduit principalement par (1) une définition formelle des bigraphes représentant le côté *back-end* d'un système Cloud qu'on appellera environnement d'hébergement (*hosting environment*), ainsi que (2) la logique de typage (*sorting*) qui introduit une axiomatisation et une sémantique robustes pour cette modélisation. Dans un deuxième temps, nous recourrons aux règles de réaction des BRS pour définir plusieurs actions d'adaptation (en termes de (dés)allocation de ressources) aux niveau infrastructure et application. À l'issue de cette étape, le but est de définir des stratégies d'élasticité réactives et opérant à plusieurs niveaux (*cross-layer*) des systèmes Cloud modélisés. Ces stratégies servent à décrire la logique qui caractérise le comportement autonomique du contrôleur d'élasticité dans sa gestion des comportements élastiques liés au système Cloud administré. Par conséquent, cette étape adresse le premier objectif de recherche (OR1).

Étape d'encodage. À partir des résultats de la première étape, c'est-à-dire la modélisation formelle de la structure et du comportement élastique d'un système Cloud, cette étape consiste à encoder et enrichir les différentes définitions basées BRS dans le langage de spécification formelle Maude. La structure des systèmes Cloud proposée est exprimée dans Maude à l'aide d'un module fonctionnel et son comportement est traduit dans un module système en termes de règles de réécriture. La spécification dans le langage Maude permet d'encoder les spécifications bigraphiques de manière à préserver leur sémantique d'une part, tout en permettant de les enrichir d'un autre. Notamment en encodant des informations quantitatives sur les entités du système et en spécifiant des règles de réécriture conditionnelles de manière à intégrer la quantification universelle et existentielle sur ces entités. Ceci permet de contrôler le comportement complexe d'un système Cloud en raisonnant sur son état global, lui-même défini à partir de l'état des entités le composant. L'objectif de cette étape est d'assurer une exécution autonomique des comportements définis à l'aide du système de réécriture exécutable Maude. L'étape d'encodage permet ainsi d'adresser le deuxième objectif de recherche (OR2).

5.2.2 Exécution et vérification de l'élasticité

Cette phase consiste à effectuer une vérification formelle des comportements élastiques définis en vue de s'assurer le leur bon fonctionnement. Il s'agit de vérifier que les actions d'élasticité exécutées de manière autonomique sont bien définies, qu'elles sont correctes et qu'elles décrivent un comportement dit « désirable ». Cela revient à définir un certain nombre de propriétés *fonctionnelles* et à identifier certaines propriétés *non-fonctionnelles* que le système se doit de satisfaire et garantir tout au long de son fonctionnement. Concrètement, le comportement du contrôleur d'élasticité doit permettre (1) de dynamiquement (re)dimensionner les ressources déployées en réponse aux changements constatés au niveau de la charge de travail, tout en garantissant (2) la stabilité et la fiabilité du système en tout temps. Le premier point consiste à montrer que les propriétés fonctionnelles inhérentes à l'élasticité sont assurées telles le dimensionnement horizontal/vertical et l'équilibrage de charge (*scale-out/in, scale-up/down* et *load-balancing*), l'absence de boucles d'adaptation (*ressource thrashing*) ou encore la non-plasticité (*non-plasticit*y). Le deuxième point consiste à s'assurer que les propriétés non-fonctionnelles du système sont

préservées telles que la sûreté (càd. *Rien de mal ne va se produire*) ou encore la vivacité (càd. *Quelque chose de bien finira par arriver*). Dans la pratique, la phase de vérification qualitative repose sur une technique de *model-checking* à base d'états (*state-based model-checking*) [Souri et al., 2018, Baier and Katoen, 2008] prise en charge par le *model-checker* du système Maude et la logique temporelle linéaire (LTL). Dans un premier temps, nous définissons un module de vérification de propriétés dans Maude où l'on va décrire de manière formelle les propriétés fonctionnelles désirées. Ensuite, nous définissons (dans le même module) une structure de *Kripke* qui décrit une sémantique dans LTL pour les propriétés définies (voir Section 4.3). D'un côté, la structure de *Kripke* permet de représenter la dynamique du système par le biais de systèmes de transitions labellisés (*labelled transition systems* - LTS) où les nœuds représentent les différents états du système et où les transitions représentent les différentes actions d'adaptations déclenchées par le contrôleur d'élasticité. D'un autre côté, le *model-checker* de Maude permet d'exécuter le modèle défini et d'en vérifier les comportements : il permet en outre d'afficher des contre-exemples montrant l'évolution de l'état du système (à partir d'un état initial donné) où certaines propriétés auraient été violées. Le *model-checker* de Maude permet de vérifier les propriétés *fonctionnelles* liées aux comportements élastiques définis, tandis la représentation à l'aide des LTS permet d'en vérifier les propriétés *non-fonctionnelles* (sûreté, vivacité). Ainsi, la phase de vérification qualitative répond au troisième objectif de recherche (OR3).

En résumé, l'association des BRS et Maude permet, d'un côté, d'apporter une formalisation forte et robuste d'un système Cloud tout en décrivant sa structure et son comportement élastique intrinsèque (Phase 1). D'un autre côté, les outils d'analyse formelle de Maude permettent d'assurer une conception correcte de ces comportements (Phase 2).

5.2.3 Évaluation de l'élasticité

La vérification d'un système revient à s'assurer qu'il se comporte bien comme prévu durant la phase de conception, tandis que sa validation équivaut à s'assurer de son adéquation avec le monde réel. En d'autres termes, la vérification et la validation peuvent être vues comme les réponses aux questions : « Avons-nous bien construit le modèle ? » et « Avons-nous construit un bon modèle ? » respectivement [Phillips, 2004]. Si la première question est adressée au niveau de la phase de vérification qualitative du modèle, la phase de validation quantitative a pour but de répondre à la deuxième.

Afin de valider le modèle proposé et d'en évaluer les performances, nous recourrons à la théorie des files d'attentes comme support de simulation et de validation. Comme illustré dans la Figure 5.2, simuler le comportement d'un système donné consiste au moins en les étapes suivantes :

- Définir des objectifs, ce qui se traduit par la formulation d'un problème et l'identification des objectifs guidant la mise au point du modèle initial.
- Collecter des données (à partir du monde réel) et les traduire en une forme injectable et compréhensible dans le modèle.
- Conduire des exécutions expérimentales du modèle à partir de ces données et analyser les résultats obtenus.
- Valider et/ou corriger le modèle selon les résultats obtenus.

Nous recourrons à la théorie des files d'attente au niveau de (1) la construction des données qui servent à la simulation, et de (2) l'analyse des résultats. Concrètement, au

lieu de proposer des données aléatoires et arbitraires, nous modélisons la dynamique environnementale du système par le biais les processus d'arrivée et de service ainsi que le modèle avec nombre variable de serveurs (voir Section 4.4).

Enfin, nous confrontons les résultats obtenus à ceux donnés par la formule d'Erlang-C [Chan, 2014, Firdhous et al., 2011] qui donne le nombre minimal et optimal de ressources à déployer afin d'assurer un niveau de service donné à partir d'un taux d'arrivée λ et d'un taux de service μ. Cela sert à mesurer les résultats obtenus à un idéal reconnu modélisant au mieux un objectif concret à atteindre, afin de repenser la conception du modèle ou, au contraire, la valider.

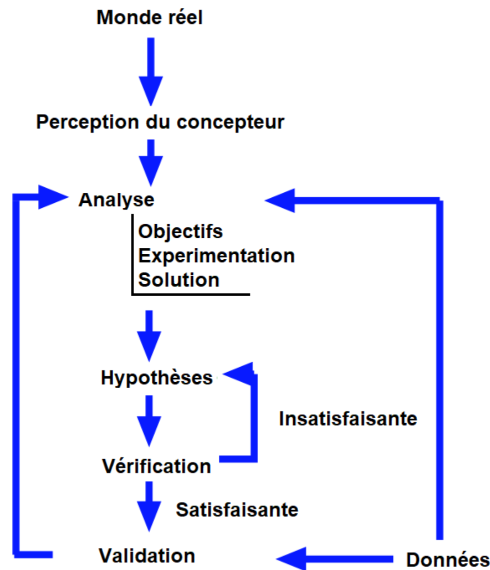


FIGURE 5.2 – Modélisation et validation

Dans la pratique, nous simulons les comportements élastiques définis avec plusieurs modèles (*patterns*) de données en entrée et plusieurs comportements à adopter. Nous étudions les résultats du dimensionnement élastique en multi-couches des systèmes Cloud afin d'en observer les politiques de haut niveau satisfaites et d'en valider les performances. Nous simulons le système avec plusieurs combinaisons de valeurs décrivant le *processus d'arrivée*, le *processus de service* et plusieurs stratégies d'élasticité à appliquer. Le processus d'arrivée indique le nombre moyen de requêtes qui arrivent dans le système et le processus de service donne une estimation des requêtes traitées qui quittent le système. Quant aux différentes stratégies applicables, elles définissent un comportement multi-couches spécifique à adopter aux niveaux infrastructure et application du système dans le but d'assurer certaines politiques de haut niveau telles que *maximiser les performances, minimiser les coûts* ou encore assurer un compromis entre les deux. Ainsi, la phase de validation quantitative répond au quatrième objectif de recherche (OR4).

5.3 Organisation des contributions

Les contributions présentées dans ce manuscrit sont organisées selon trois Chapitres où nous détaillons chaque phase notre solution pour la spécification, la vérification et l'évaluation de l'élasticité multi-couches dans le Cloud. Nous y présentons les différentes contributions réalisées au cours de nos travaux de thèse. Le Chapitre 6 couvre la première

étape (modélisation) de la phase de spécification formelle. Le Chapitre 7 aborde la deuxième étape (encodage) de la phase de spécification formelle ainsi que la totalité de la phase de vérification formelle de l'élasticité. Enfin, le dernier Chapitre 8 traitera de la troisième phase de notre solution, à savoir l'évaluation quantitative de l'élasticité.

Chapitre 6

Formalisation de l'élasticité multi-couches dans le Cloud

Sommaire

6.1 Introduction 82
6.2 Approche bigraphique pour la modélisation des systèmes Cloud élastiques . 83
6.2.1 Méta-modèle d'un système Cloud élastique 84
6.2.2 Sémantique bigraphique pour la structure d'un système Cloud 85
6.2.3 Sémantique basée BRS pour la dynamique d'un système Cloud élastique 88
6.2.4 Bilan et problématique 93
6.3 Stratégies d'élasticité multi-couches dans le Cloud 94
6.3.1 Dimensionnement Horizontal 97
6.3.2 Migration et Load Balancing 99
6.3.3 Dimensionnement Vertical 100
6.3.4 Bilan 102
6.4 Conclusion 102

6.1 Introduction

La gestion de l'élasticité d'un système Cloud est une tâche complexe qui dépend de plusieurs facteurs qui se chevauchent tels que les variations de la charge de travail, la quantité des ressources disponibles au niveau de l'infrastructure Cloud sous-jacente et des services déployés, l'état actuel du système ou encore, la logique qui contrôle son comportement dit "élastique", caractérisant l'allocation dynamique des ressources.

Le comportement élastique d'un système Cloud est généralement géré par un contrôleur d'élasticité, une entité autonomique qui décide des différentes actions à déclencher afin de garantir une gestion efficace de l'élasticité. Cela revient à redimensionner le système en ajoutant et en retirant des ressources informatiques de manière à assurer une bonne qualité de service, tout en minimisant les coûts liés à l'utilisation de ces ressources.

La modélisation d'un tel comportement est cruciale afin de gérer efficacement l'élasticité d'un système Cloud. En premier lieu, cela consiste à identifier les différentes actions de redimensionnement des ressources pouvant être déclenchées par le contrôleur d'élasticité. Et, en second lieu, il s'agit de décrire comment ces actions sont déclenchées de manière à garantir des comportements élastiques corrects.

En vue de sa grande complexité, le comportement élastique d'un système Cloud peut entrainer des situations indésirables telles que l'instabilité dans l'allocation des ressources

et la dégradation sévère de la qualité de service et de la fiabilité du système. Ainsi, il est également nécessaire de définir un modèle permettant de décrire précisément la structure des systèmes Cloud dont l'élasticité est contrôlée. Un tel modèle sert à déterminer les changements potentiels pouvant être appliqués au niveau d'une architecture Cloud, et d'en restreindre le champ de possibilités, afin d'assurer une dynamique correcte et prévenir les comportements non désirables. Dans cette thèse, nous estimons que les méthodes formelles, caractérisées par leur efficacité, fiabilité et précision, incarnent une excellente solution pour réduire et maitriser la complexité liée à la modélisation des systèmes Cloud, et de leur comportement élastique.

Dans ce Chapitre, nous présentons une approche de modélisation formelle afin de spécifier et contrôler le comportement élastique d'un système Cloud. Nous y présentons un modèle à base du formalisme des Systèmes Réactifs Bigraphiques (BRS) permettant de modéliser les structures et les comportements de systèmes Cloud. Dans la Section 6.2, nous définissons une sémantique bigraphique accompagnée d'une logique de typage permettant de représenter les architectures Cloud avec un niveau de détails considérable, permettant d'identifier et de modéliser les différentes entités constituant ces architectures. Nous y définissons également un ensemble de règles de réaction bigrapiques permettant de décrire les actions de reconfiguration applicables sur une architecture Cloud, au niveau des couches infrastructure et application de celui-ci. La Section 6.3 présente certaines de nos principales contributions. Nous y proposons une modélisation des comportements autonomiques du contrôleur d'élasticité dans le Cloud. Précisément, nous y définissons un ensemble de stratégies d'élasticité (horizontale, verticale, migration et load-balancing). Ces stratégies décrivent des comportements autonomiques réactifs pour la gestion des ressources Cloud en multi-couches, c'est-à-dire aux niveaux infrastructure et application.

6.2 Approche bigraphique pour la modélisation des systèmes Cloud élastiques

Dans cette Section, nous présentons un modèle formel basé sur les systèmes réactifs bigraphique (BRS) et leur logique de typage (*sorting*) permettant de modéliser les aspects architecturaux des systèmes Cloud ainsi que leur comportement élastique. Dans un premier temps, nous introduisons les différents éléments architecturaux et comportementaux constituant ces systèmes. Ensuite, nous présentons une sémantique formelle basée sur les BRS permettant de les représenter formellement.

La partie *front-end*. Représente l'environnement extérieur au système Cloud et sert à identifier les sollicitations qu'il peut en recevoir. Ces sollicitations prennent la forme de requêtes envoyées par les clients ou utilisateurs finaux des services déployés au sein du système.

La partie *back-end*. Représente l'environnement d'hébergement (*hosting environment*) d'un système Cloud. Elle regroupe toutes les ressources (matérielles et logicielles) qui y sont déployées. Ces ressources sont les différentes entités de calcul : serveurs physiques (*server*), machines virtuelles (*VM*), applications (ou instances de *services*), ainsi que les différentes ressources informatiques disponibles : principalement CPU et mémoire RAM. L'environnement d'hébergement montre la relation d'imbrication (hébergement) que peuvent avoir les différentes entités entre elles. Une requête provenant du front-end est prise en charge par une instance de service. Cette instance de service est hébergée par une VM, qui est elle-même hébergée au sein d'un serveur physique. La modélisation de

la partie back-end permet également de décrire le déploiement (ou offre) des ressources informatiques au sein des serveurs (*offering*) et leur virtualisation au niveau des VMs. Chaque entité (serveur, VM, instance de service) dispose de seuil d'hébergement maximal et minimal (*max/min capacity*) afin d'en déterminer l'état d'usage ou de dimensionnement : stable, non-utilisé (*idle*), ou sur/sous-dimensionné (*over/under-provisionned*).

6.2.1 Méta-modèle d'un système Cloud élastique

Nous présentons l'ensemble des notions, concepts et entités constituant un système Cloud élastique à travers un méta-modèle (voir Figure 6.1). Le méta modèle proposé met l'accent sur une vision orientée sur l'interaction du système Cloud côté back-end avec son interface (*front-end*), son contrôleur d'élasticité (*elasticity controller*) et sa partie administration (*management*).

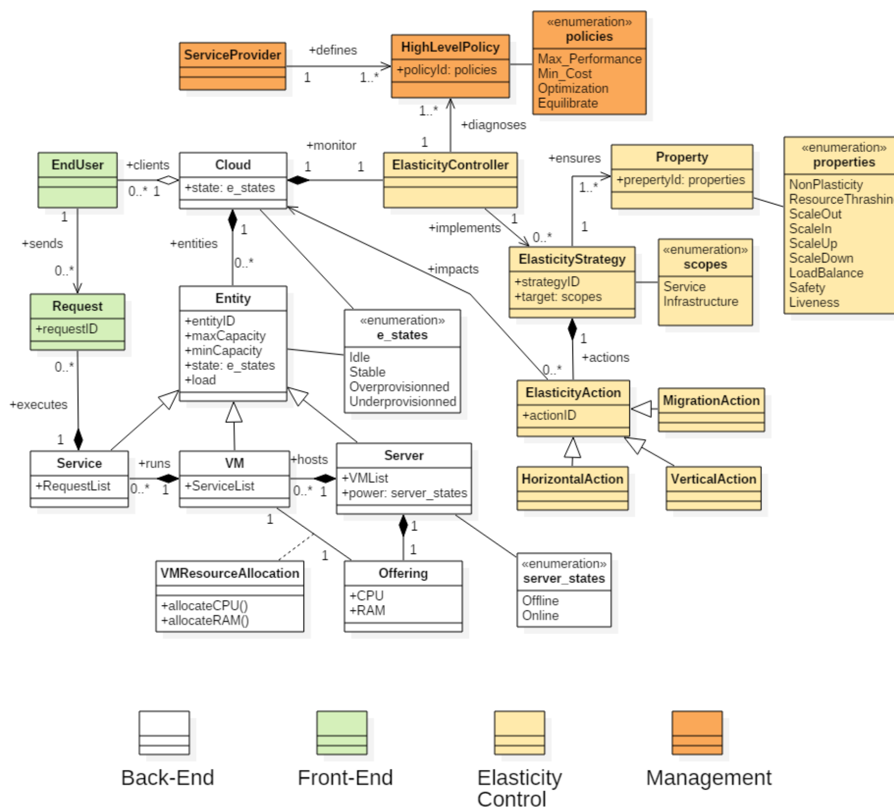


FIGURE 6.1 – Meta modèle : vision d'ensemble d'un système Cloud élastique

Le contrôleur d'élasticité (*elasticity controller*). Surveille périodiquement l'environnement d'hébergement pour en déterminer l'état en termes de ressources disponibles (sur/sous-dimensionnement – *over/under-provisionning*). Cela revient à déterminer l'état des différentes entités déployées afin d'identifier et d'appliquer les actions adéquates pour stabiliser l'état du système. À partir de ses observations (*monitoring*), le contrôleur d'élasticité opère selon différentes stratégies en appliquant des actions d'adaptation (dimensionnement) selon plusieurs méthodes (*horizontal, vertical, migration*), aux niveaux *infrastructure* (serveurs, VMs) et *application* (instances de service, requêtes). En plus du contrôle des différentes actions d'adaptation en vue d'atteindre un état d'élasticité *stable* (càd. Où aucune action de redimensionnement ne serait demandée), le contrôleur d'élasticité se doit de garantir certaines propriétés et politiques de haut niveau liées au compor-

tement qu'il décrit. Les propriétés *fonctionnelles* et *non-fonctionnelles* sont inhérentes aux comportements induits par le contrôleur d'élasticité. Les propriétés fonctionnelles définissent le comportement de dimensionnement dynamique des ressources (*scale-out/in*, *scale-up/down*, *load-balancing*, etc.), et les propriétés non-fonctionnelles concernent la *sûreté* et la *vivacité* du système dans son comportement élastique.
La partie administration (*managment*). Spécifie l'ensemble des politiques de haut niveau (*high-level policies*) que le contrôleur d'élasticité tente de garantir tout au long son fonctionnement. Les politiques de haut niveau définissent les différentes exigences que le fournisseur du service Cloud (*service provider*) souhaite appliquer sur son produit (e.g. *minimiser les coûts, maximiser les performances*, etc.).

6.2.2 Sémantique bigraphique pour la structure d'un système Cloud

Structurellement, nous modélisons un système Cloud par un bigraphe CS où nous nous concentrons sur les éléments constituant son côté *back-end*. Dans notre approche de modélisation, le système Cloud est représenté par une région et chaque entité structurelle le composant est représentée par un nœud. Les connexions entre les différents nœuds ainsi qu'avec l'environnement extérieur sont représentés par des hyper-arcs.

Définition 21 . Le bigraphe CS est donné formellement par :

$$CS = (V_{CS}, E_{CS}, ctrl_{CS}, CS^P, CS^L) : I_{CS} \rightarrow J_{CS}$$

— V_{CS} est un ensemble fini de nœuds représentant les différentes entités (service, VM, serveur, etc.) composant le système Cloud.

— E_{CS} est un ensemble fini d'hyper-arcs représentant les différentes connexions pouvant lier les entités du Cloud entre elles.

— $ctrl_{CS} : V_{CS} \rightarrow \mathcal{K}$ est une fonction de transformation qui associe à chaque nœud $v_i \in V_{CS}$ un contrôle $c \in K$ indiquant le nombre de ports. La signature $\mathcal{K}$ est un ensemble fini de contrôles associés aux éléments du système Cloud.

— $CS^P = (V_{CS}, ctrl_{CS}, prnt_{CS}) : m \rightarrow n$ est le graphe de places associé à CS. m et n représentent le nombre de sites et le nombre de régions. $prnt_{CS} : m \uplus V_{CS} \rightarrow V_{CS} \uplus n$ est une fonction de parentalité qui associe à chaque entité son parent hiérarchique (e.g. le parent d'un nœud VM est un nœud serveur).

— $CS^L = (V_{CS}, E_{CS}, ctrl_{CS}, link_{CS}) : X \rightarrow Y$ est le graphe de liens de CS, où $link_{CS} : X \uplus P \rightarrow E_{CS} \uplus Y$ est une fonction de transformation qui spécifie les interactions de chaque entité du Cloud. X, Y et P représente respectivement l'ensemble de noms internes, l'ensemble de noms externes et l'ensemble de ports de CS.

— $I_{CS} =< m, X >$ et $J_{CS} =< n, Y >$ représentent respectivement les interfaces internes et externes du bigraphe CS.

Afin d'en définir une sémantique précise et rigoureuse, nous associons au bigraphe CS une discipline de typage (*sorting*) Σ_{CS}.

Définition 22 . La discipline de typage associée au bigraphe CS modélisant un système Cloud élastique est définie par le triplet $\Sigma_{CS} = \{\Theta_{CS}, \mathcal{K}_{CS}, \Phi_{CS}\}$. Où Θ_{CS} représente un ensemble non-vide de sortes de CS, $\mathcal{K}_{CS}$ est une signature Σ_{CS}-typée qui associe une sorte à chaque contrôle de CS et Φ_{CS} est un ensemble non-vide de règles de formation imposant des restrictions de construction pour CS.

Les sortes et les contrôles utilisés pour représenter formellement les différents éléments du bigraphe CS sont indiqués dans le Tableau 6.1. Nous utilisons la notation disjonctive pour indiquer qu'un nœud peut être de plusieurs sortes. Par exemple, un nœud noté $\widehat{ab}$ peut avoir une sorte a ou une sorte b.

TABLE 6.1 – Contrôles et sortes du bigraphe CS

Description	Contrôle	Arité	Sorte	Notation graphique
Machine Physique				
Serveur	SE	3	e	Rectangle
Serveur surchargé	SE^O			
Serveur non-utilisé	SE^I			
Pool de ressources disponibles	RD	1	r	Rectangle
Virtualisation				
Machine virtuelle	VM	3	v	Rectangle
Machine virtuelle surchargée	VM^O			
Machine virtuelle sur-dimensionnée	VM^P			
Machine virtuelle non-utilisée	VM^I			
Pool de ressources allouées	RV	1	r	Rectangle
Application				
Instance de service	S	1	s	Cercle
Instance de service surchargée	S^O			
Instance de service non-utilisée	S^I			
Requête	R	0	q	Triangle
Ressources Informatiques				
Unité CPU	CU	0	c	Cercle
Unité de mémoire RAM	M	0	m	Cercle

Pour modéliser la partie *back-end* ou environnement d'hébergement (*hosting environment*) d'un système Cloud, nous définissons un ensemble de sortes $\Theta_{CS} = \{e, v, s, q, r, c, m\}$ associées à différents contrôles suivant la signature $\mathcal{K}_{CS}$:

$$\mathcal{K}_{CS} = \left\{ \begin{array}{rl} e: & \{SE, SE^O, SE^I\}, \\ v: & \{VM, VM^O, VM^P, VM^I\}, \\ s: & \{S, S^O, S^I\}, \\ r: & \{RD, RV\}, \\ q: & \{R\}, \\ c: & \{CU\}, \\ m: & \{M\} \end{array} \right\}$$

TABLE 6.2 – Règles de construction Φ_{CS} du bigraphe CS

	Description de la règle
Φ_0	Tous les fils d'une 0-région (*back-end*) sont de sorte e
Φ_1	Tous les fils d'un e-nœud sont de sorte $\widehat{vr}$
Φ_2	Tous les fils d'un v-nœud sont de sorte $\widehat{sr}$
Φ_3	Tous les fils d'un s-nœud sont de q-sorte
Φ_4	Tous les fils d'un r-nœud sont de sorte $\widehat{cm}$
Φ_5	Tous les nœuds de sorte $\widehat{qcm}$ sont atomiques
Φ_6	Tous les nœuds de sorte $\widehat{evsqcm}$ sont actifs et les r-nœuds sont passifs
Φ_7	Dans un e-nœud, un port est toujours lié à un w-nom, un port peut être lié aux v-nœuds fils et un port est lié au r-nœud fils
Φ_8	Dans un v-nœud, un port est toujours lié au r-nœud fils, un port est toujours lié au e-nœud parent et un port peut être lié aux s-nœuds fils
Φ_9	Dans un s-nœud, le port est toujours lié au v-nœud parent
Φ_{10}	Dans un r-nœud, le port est toujours lié au $\widehat{ev}$-nœud parent

Les différentes sortes utilisées sont associées aux contrôles décrivant les différents nœuds composant le système. Les nœuds de même sorte peuvent avoir des signatures différentes afin d'indiquer leur état ou leur rôle. Un serveur ou une instance de service (serveur/service) peut avoir plusieurs états : surchargé (*Overloaded* : SE^O/S^I), non-utilisé (*Idle* : SE^I/S^I) ou stable (SE/S). Ainsi, un nœud de sorte e/s représente forcément un serveur/service mais peut avoir plusieurs contrôles pour indiquer son état. De la même manière, une machine virtuelle est forcément de sorte v mais peut avoir plusieurs états indiqués par son contrôle : sous-dimensionnée ou surchargée (*overloaded* : VM^O), sur-dimensionnée (*Over-provisionned* : VM^P), non-utilisée (*Idle* : VM^I) ou stable (VM). Les pools de ressources ont la même structure et donc la même sorte r. Cependant, on distingue les pools des ressources disponibles au niveau d'un serveur et ceux alloués à une machine virtuelle par les deux contrôles RD et RV respectivement. Les ressources (unités de CPU ou de mémoire RAM) sont associées aux sortes c et m, et sont de contrôles CU et M respectivement. La signature associée à chaque contrôle lui définit également un

nombre de ports (e.g. un serveur dispose de 3 ports) et une notation graphique donnée (e.g. une instance de service est représentée par un cercle). Les règles de construction Φ_{CS}, prennent la forme $\Phi_i, i \in [0..10]$ et permettent de restreindre les possibilités de construction du bigraphe CS. Le but est de garantir une conception de bigraphes représentant uniquement des structures correctes d'un système Cloud (voir Tableau 6.2).

Les règles de construction Φ_{CS} permettent d'imposer des contraintes structurelles afin de garantir des modélisations correctes autorisées par le bigraphe CS. Les règles $\Phi_0 - \Phi_5$ définissent les contraintes sur l'imbrication hiérarchique des différentes entités, la règle Φ_6 décrit leur activité et les règles $\Phi_7 - \Phi_{10}$ définissent les restrictions sur leurs liens.

Par exemple, la règle Φ_0 stipule que la région principale, notée 0 et qui représente le système Cloud ne peut avoir de fils que des nœuds de sorte e (serveurs). La règle Φ_6 précise que les nœuds représentant les pools de ressources sont passifs (ne prennent part à aucune règle de réaction) tandis que tous les autres nœuds sont actifs. Enfin, la règle Φ_7 impose, entre autres, que tous les serveurs soient liés au nom externe w pour *workload* qui représente l'interface de connexion avec la partie *front-end* du système Cloud.

Exemple. Considérons un système Cloud quelconque où un serveur physique est déployé et où une machine virtuelle est lancée. La VM héberge deux instances de service en exécution qui prennent en charge une requête chacune. Au niveau de l'allocation des ressources, le serveur physique dispose de deux unités de CPU et de deux unités de mémoire RAM disponibles. Quant à la VM lancée, son allocation de ressources est d'une unité de CPU et d'une unité de mémoire RAM. Un tel système peut être représenté par un bigraphe CS selon la sémantique structurelle bigraphique introduite (voir Figure 6.2). Le bigraphe CS est donné par une interface $< 0, \emptyset > \rightarrow < 1, w >$ indiquant qu'il ne dispose d'aucun site, qu'il n'a pas de nom interne, qu'il dispose d'une seule région et qu'il a un nom externe w.

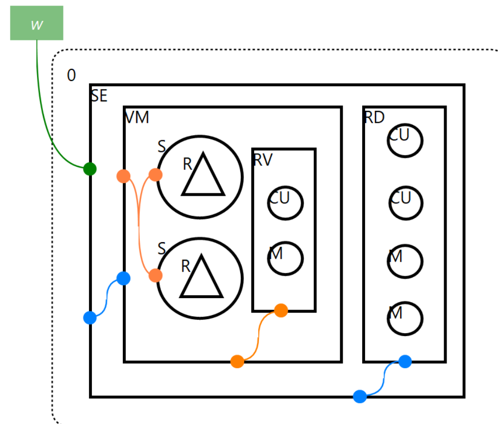


FIGURE 6.2 – Exemple d'un bigraphe CS modélisant un système Cloud

6.2.3 Sémantique basée BRS pour la dynamique d'un système Cloud élastique

Le comportement élastique des système Cloud est une tâche complexe qui dépend de plusieurs facteurs qui se chevauchent tels que la variation de la demande, les ressources disponibles ou encore la logique qui régit le comportement du contrôleur d'élasticité dans sa tâche de redimensionnement dynamique des ressources. La modélisation d'un tel comportement est cruciale afin de gérer efficacement l'élasticité d'un système Cloud. Cela

consiste à (1) identifier les différentes actions de redimensionnement des ressources pouvant être déclenchées par le contrôleur d'élasticité, et (2) décrire comment ces actions sont déclenchées de manière à garantir des comportements élastiques corrects et désirables.

En se basant sur la sémantique bigraphique précédemment définie pour la modélisation des aspects structurels des systèmes Cloud, nous introduisons ici une sémantique basée sur les systèmes réactifs bigraphique afin d'en spécifier les aspects dynamiques, décrivant leur élasticité. Dans un premier temps, nous définissons un ensemble de règles de réaction bigraphiques décrivant les différentes actions de dimensionnement et de migration de ressources et entités à différents niveaux du système Cloud (serveur, VM, instance de service et requête). Dans la Section suivante, nous expliquerons comment les règles de réaction spécifiées sont déclenchées par le contrôleur d'élasticité. Cela consiste en plusieurs stratégies d'élasticité décrivant la logique qui régit la gestion autonomique de l'élasticité dans un système Cloud.

Les règles de réaction définies ici décrivent les différents mécanismes appliqués à différents niveaux du système Cloud, afin d'en gérer l'élasticité. Il s'agit de modéliser les actions relatives à l'augmentation, la diminution et la migration des différentes ressources et entités déployées dans le système. Le Tableau 6.3 regroupe les règles de réaction définies. Elles prennent la forme $R_i = R \rightarrow R'$, où i est l'indice de la règle, R est la partie redex de la réaction et R' est sa partie reactum. Concrètement, les règles décrivent les différentes actions liées au dimensionnement ou élasticité horizontale (*scale-out/in* : $R1 - R6$), à l'élasticité verticale (*scale-up/down* : $R7 - R10$), à la migration ($R11, R12$) et à l'équilibrage de charge (*load-balancing* : $R13, R14$) aux niveaux infrastructure et application d'un système Cloud (voir Figure 6.3).

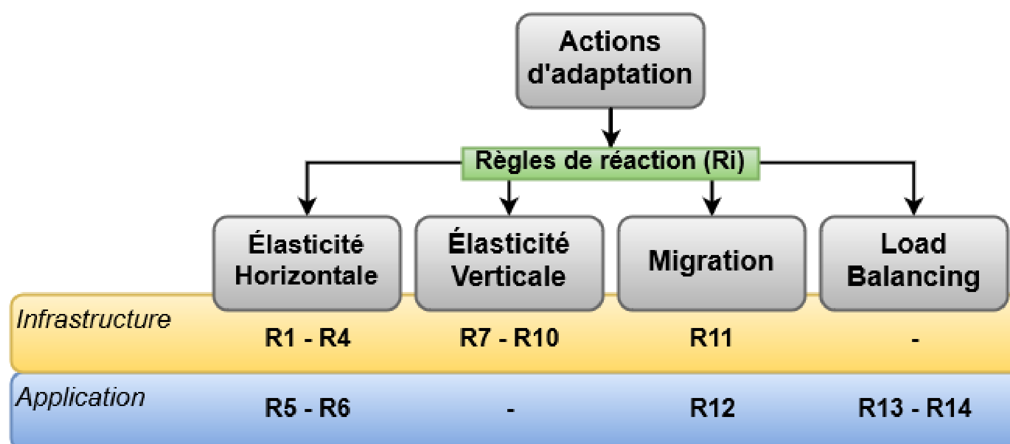


FIGURE 6.3 – Modélisation des actions d'adaptations élastiques par des règles de réaction bigraphiques

Les règles spécifiées décrivent des comportements de base pouvant être appliqués sur le système Cloud comme l'ajout/retrait des ressources Cloud au niveau infrastructure (Serveur, VM, CPU, RAM) et au niveau application (instances de service et requêtes). On représente également les actions de migration d'une instance de VM/service vers un autre serveur/VM ou encore les actions de load balancing via la redirection d'une requête vers une autre instance de service. Nous rappelons que les règles de réaction spécifiées s'appliquent dans le contexte de la sémantique structurelle précédemment introduite (bigraphe CS). Ainsi, les réactions concernent les différentes entités d'un système Cloud représentées par des nœuds les modélisent sémantiquement via des contrôles et des sortes associées.

TABLE 6.3 – Règles de réaction pour le dimensionnement et la migration des ressources

Élasticité horizontale (Scale-Out/In)	
Infrastructure	Mise en service d'un serveur physique
	$R1 \stackrel{\text{def}}{=} id \rightarrow (SE.d1) \mid id$
	Extinction d'un serveur physique
	$R2 \stackrel{\text{def}}{=} (SE^I.d1) \mid id \rightarrow id$
	Déploiement d'une nouvelle instance de machine virtuelle
	$R3 \stackrel{\text{def}}{=} (SE.d1) \mid id \rightarrow (SE.(VM.d2) \mid d1) \mid id$
	Consolidation d'une instance de machine virtuelle
	$R4 \stackrel{\text{def}}{=} (SE.(VM^I.d2) \mid d1) \mid id \rightarrow (SE.d1) \mid id$
Application	Déploiement d'une nouvelle instance de service
	$R5 \stackrel{\text{def}}{=} (SE.(VM.d2) \mid d1) \mid id \rightarrow (SE.(VM.(S.d3) \mid d2) \mid d1) \mid id$
	Consolidation d'une instance de service
	$R6 \stackrel{\text{def}}{=} (SE.(VM.(S^I.d3) \mid d2) \mid d1) \mid id \rightarrow (SE.(VM.d2) \mid d1) \mid id$
Élasticité verticale (Scale-Up/Down)	
Infrastructure	Ajout d'une unité de CPU à une VM
	$R7 \stackrel{\text{def}}{=} SE.((RD.CU \mid d4) \mid (VM.(RV.d3) \mid d2) \mid d1) \mid id$ $\rightarrow SE.((RD.d4) \mid (VM.(RV.CU \mid d3) \mid d2) \mid d1) \mid id$
	Retrait d'une unité de CPU d'une VM
	$R8 \stackrel{\text{def}}{=} SE.((RD.d4) \mid (VM.(RV.CU \mid d3) \mid d2) \mid d1) \mid id$ $\rightarrow SE.((RD.CU \mid d4) \mid (VM.(RV.d3) \mid d2) \mid d1) \mid id$
	Ajout d'une unité de mémoire RAM à une VM
	$R9 \stackrel{\text{def}}{=} SE.((RD.M \mid d4) \mid (VM.(RV.d3) \mid d2) \mid d1) \mid id$ $\rightarrow SE.((RD.d4) \mid (VM.(RV.M \mid d3) \mid d2) \mid d1) \mid id$
	Retrait d'une unité de mémoire RAM d'une VM
	$R10 \stackrel{\text{def}}{=} SE.((RD.d4) \mid (VM.(RV.M \mid d3) \mid d2) \mid d1) \mid id$ $\rightarrow SE.((RD.M \mid d4) \mid (VM.(RV.d3) \mid d2) \mid d1) \mid id$
Migration	
Infrastructure	Migration d'une VM vers un autre serveur
	$R11 \stackrel{\text{def}}{=} (SE.((VM.d3) \mid d2)) \mid (SE.d1) \mid id \rightarrow (SE.d2) \mid (SE.((VM.d3) \mid d1)) \mid id$
Application	Migration d'une instance de service vers une autre VM
	$R12 \stackrel{\text{def}}{=} SE.(((VM.(S.d4) \mid d3) \mid (VM.d2)) \mid d1) \mid id$ $\rightarrow SE.(((VM \mid d2) \mid (VM.(S.d4))) \mid d1) \mid id$
Load Balancing	
Application	Redirection d'une requête vers une instance de service d'une autre VM
	$R13 \stackrel{\text{def}}{=} SE.((VM.(S.q \mid d5) \mid d3) \mid (VM.(S.d4) \mid d2) \mid d1) \mid id$ $\rightarrow SE.((VM.(S.d5) \mid d3) \mid (VM.(S.q \mid d4) \mid d2) \mid d1) \mid id$
	Redirection d'une requête vers une autre instance de service de la même VM
	$R14 \stackrel{\text{def}}{=} SE.((VM.(S.R \mid d4) \mid (S.d3) \mid d2) \mid d1) \mid id$ $\rightarrow SE.((VM.(S.d4) \mid (S.R \mid d3) \mid d2) \mid d1) \mid id$

Actions de l'élasticité horizontale. Ces actions représentent le comportement de dimensionnement horizontal (*Scale-Out/In*) en rajoutant/retirant un serveur ($R1, R2$) ou une

VM ($R3, R4$) au niveau infrastructure et une instance de service ($R5, R6$) au niveau application. Par exemple, la règle $R3$ décrit le déploiement d'une nouvelle VM au niveau d'un serveur quelconque. Elle est donnée algébriquement par le côté gauche, le redex, qui comprend un nœud SE dont le contenu est abstrait via un site $d0$. Le *redex* se réécrit en le côté droit de la règle, ou *reactum*, où un nœud de contrôle VM est déployé et dont le contenu est abstrait via un site $d1$. L'abstraction via les sites permet de ne pas considérer certains détails de la réaction comme l'allocation des ressources dans le cas des actions de l'élasticité horizontale où la capacité d'approvisionnement est supposée théoriquement infinie (la question de l'allocation des ressources sera abordée de manière détaillée dans la partie dédiée au comportement du contrôleur d'élasticité). Les actions de retrait des ressources sont appliquées sur des entités non utilisées, de manière à éviter la perte de données. De manière générale, un serveur/VM/service ne peut être retiré que lorsqu'il n'y a aucune VM/service/requête qui y soit hébergée. Par exemple, la règle $R6$ décrit l'action de retrait d'une instance de service non utilisée (*idle*), où sa partie *redex* montre un nœud de contrôle S^I (instance de service non-utilisée) déployé au sein d'un nœud VM quelconque. Le *redex* est réécrit en un *reactum* où le nœud représentant le service non utilisé est supprimé. Les entités non concernées par cette action sont abstraites via les sites notés di où i est l'indice du site. La forme graphique des règles $R3$ et $R6$ est donnée dans la Figure 6.4.

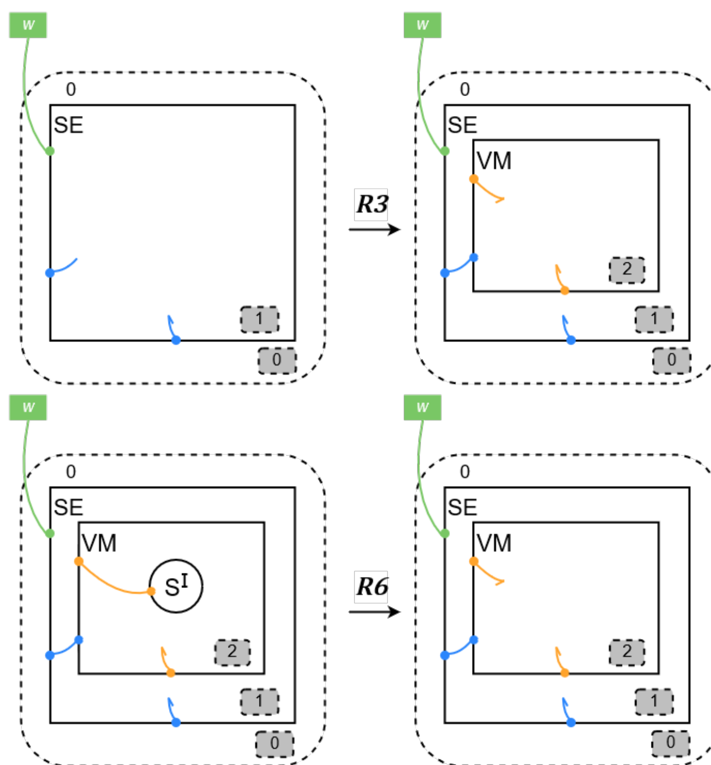


FIGURE 6.4 – Forme graphique des règles de réaction $R3$ et $R6$

Actions de l'élasticité verticale. Ces actions représentent le dimensionnement vertical (*Scale-Up/Down*) en ajoutant ($R7, R9$) et en retirant ($R8, R10$) des ressources informatiques (unités de CPU et de mémoire RAM) à une VM quelconque. Au niveau des règles $R7$ et $R9$, une unité de CPU ou de mémoire RAM (nœud de contrôle CU ou M) est déplacée d'un pool de ressources disponibles, au niveau d'un serveur, au pool de ressources allouées à la VM concernée. Algébriquement, cela consiste à réécrire le *redex* de

la règle en son côté *reactum* de manière à exprimer cette action. Les règles $R8$ et $R10$ appliquent l'inverse de ces comportements en libérant des ressources de la VM et en les replaçant au sein du pool des ressources disponibles du serveur. La forme graphique des règles $R7$ et $R10$ est donnée dans la Figure 6.5.

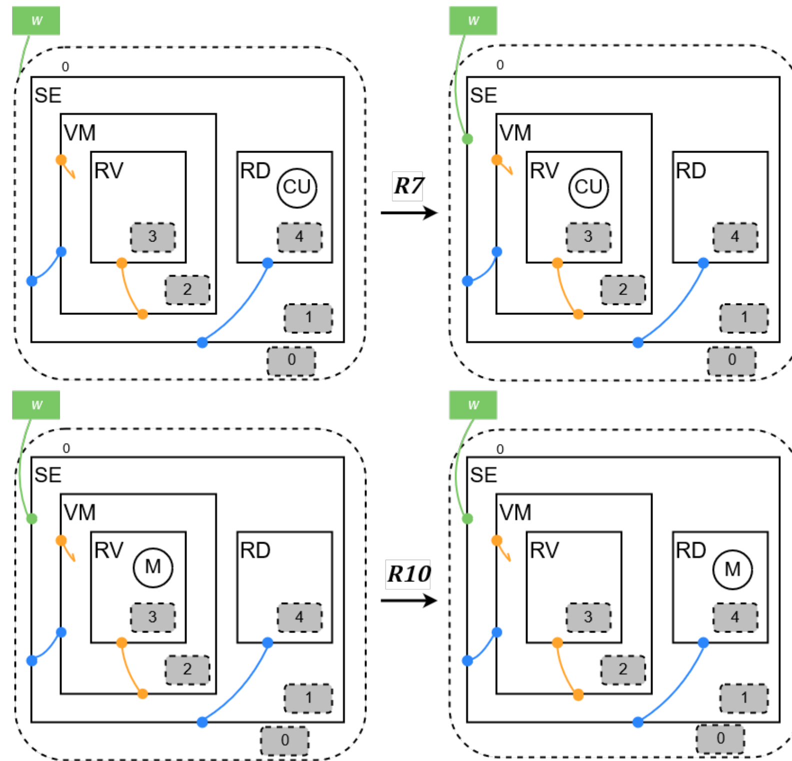


FIGURE 6.5 – Forme graphique des règles de réaction $R7$ et $R10$

Actions de migration. Les actions de migration concernent les niveaux infrastructure et application. Il s'agit respectivement de déplacer une VM d'un serveur à un autre ($R11$) ou une instance de service d'une VM à une autre ($R12$). Algébriquement, la règle $R11$ spécifie un *redex* où un nœud de contrôle VM est déployé au sein d'un nœud donné de contrôle SE. Le redex se réécrit en un *reactum* où le nœud VM est déplacé vers un autre nœud serveur, disponible au niveau de la même région. La règle $R12$ spécifie un comportement similaire mais qui concerne un nœud de contrôle S qui change de nœud VM hôte. Les formes graphiques des règles $R11$ et $R12$ sont données dans la Figure 6.6.

Actions de load balancing. L'équilibrage de charge concerne le niveau application où les requêtes gérées au niveau d'une instance de service sont redirigées vers une autre instance. Nous distinguons deux cas d'équilibrage de charge : la redirection d'une requête vers une instance de service se trouvant dans une VM distante ($R13$) ou bien vers une autre instance se trouvant dans la même VM ($R14$). Algébriquement, la règle $R13$ donne un *redex* où le nœud de contrôle R à rediriger (requête) est hébergé au sein d'un nœud de contrôle S (instance de service), lui-même dans un nœud de contrôle VM. La *redex* se réécrit en *reactum* où le nœud requête est déplacé vers un nœud service d'une autre VM déployée au sein du même serveur. La règle $R14$ accomplit la même tâche, à la différence que le nœud de contrôle S en cible de la redirection de la requête se trouve au sein de la même VM que le nœud initialement hôte. Dans ces deux représentations, les différents sites servent à abstraire les nœuds qui ne prennent pas part aux réactions souhaitées. La

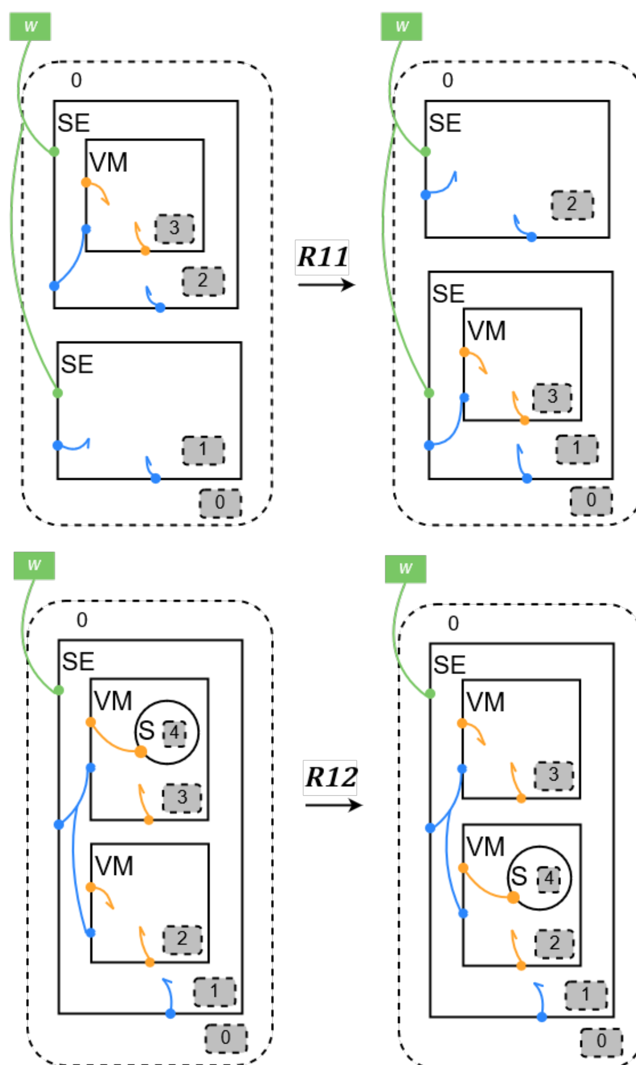


FIGURE 6.6 – Forme graphique des règles de réaction $R11$ et $R12$

forme graphique des règles $R13$ et $R14$ est donnée dans la Figure 6.7.

6.2.4 Bilan et problématique

Dans cette sous-section, nous avons identifié des règles de réaction bigraphiques afin de décrire la dynamique structurelle d'un système Cloud. L'ensemble de règles de réaction $R1 - R14$ défini sert à modéliser les différentes actions liées aux mécanismes d'élasticité (horizontale, verticale, migration) ainsi que les actions de load balancing, aux niveau infrastructure (serveur, VM) et application (instance de service, requête) d'un système Cloud donné.

Rappelons que les règles de réaction bigraphiques décrites ici sont à déclenchement dit spontané. C'est-à-dire qu'une règle est déclenchée automatiquement et inconditionnellement si la configuration (ou *pattern*) décrit par son côté *redex* survient (ou *matche*) dans le contexte du bigraphe CS. De plus, l'abstraction via les sites (di) permet de ne considérer que les entités qui prennent part à une réaction. De ce fait, une règle de réaction peut également être déclenchable si son *redex* correspond à un sous-contexte de CS. Sémantiquement, les transitions définies décrivent des réécritures structurellement correctes par définition car elles satisfont la logique de typage (*sorting*) imposée

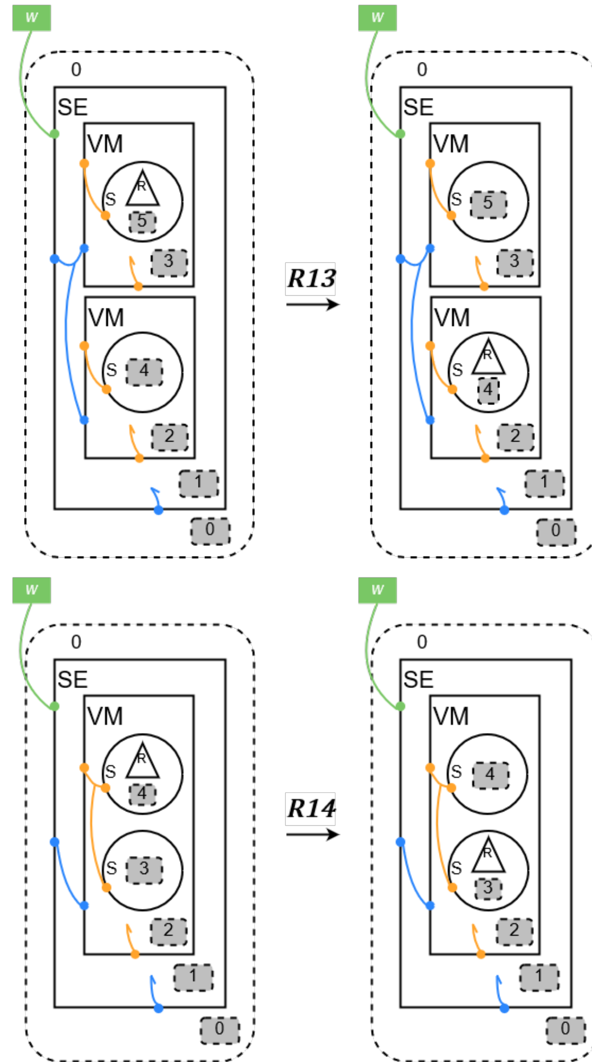


FIGURE 6.7 – Forme graphique des règles de réaction $R13$ et $R14$

sur CS. Cela assure des réécritures qui préservent les propriétés structurelles du bigraphe CS et donc son intégrité sémantique. D'un autre côté, plusieurs règles différentes peuvent être déclenchables en même temps de manière concurrente, ce qui induit une dynamique de nature non-déterministe. En d'autres termes, les règles de réaction décrivent les différentes actions *possibles* mais ne spécifient pas de raisonnement pour contrôler leur déclenchement.

Ainsi, le déclenchement inconditionnel et concurrent des différentes règles de réaction induit un comportement élastique incontrôlable et potentiellement indésirable. Pour y pallier, il est primordial d'imposer des restrictions sur l'application des règles et donc une logique contrôlant les comportements introduits par les règles de réaction. Dans la Section 6.3, nous définissons cette logique en termes de stratégies appliquées par le contrôleur d'élasticité afin de gérer les différentes actions d'adaptation d'un système Cloud.

6.3 Stratégies d'élasticité multi-couches dans le Cloud

Le contrôleur d'élasticité est un gestionnaire autonomique qui décide des différentes actions à entreprendre afin de gérer de manière dynamique et automatisées l'élasticité d'un système Cloud. Généralement, le comportement du contrôleur est décrit à tra-

vers des stratégies pouvant spécifier plusieurs méthodes d'élasticité et pouvant opérer en mutli-couches, c'est-à-dire plusieurs niveaux du système Cloud. Une stratégie décrit un comportement à suivre afin de piloter les actions d'adaptation élastiques d'un système donné. Il s'agit de spécifier une logique décrivant quand et comment les actions d'adaptation sont déclenchées. Dans cette sous-section, nous introduisons un nombre de stratégies d'élasticité réactives afin de contrôler le déclenchement des règles de réaction précédemment identifiées. Cela consiste à restreindre l'application des règles via des conditions, de manière à appliquer une règle de réaction uniquement quand cela est désiré (càd. Quand les conditions de déclenchement de cette action sont satisfaites). Concrètement, nous définissons des stratégies pour décrire les mécanismes d'élasticité horizontale, verticale, la migration et le load balancing. Une stratégie réactive prend la forme suivante :

$$Strategie \; : \; Si\, Condition(s)\, Alors\, Action(s)$$

Dans le contexte de la sémantique bigraphique précédemment introduite, une condition prend la forme $(CS \vDash \varphi i)$. Cette condition est satisfaite ssi $\exists$ un bigraphe $B_{\varphi}i$, encodant un prédicat φi, qui survient dans le contexte de CS. Un prédicat φi souvent exprimé dans la logique du premier ordre, sert à définir un état du système Cloud et $B_{\varphi i}$ définit un modèle bigraphique encodant cet état. Par exemple, le prédicat $\varphi \equiv$ « il existe une machine virtuelle surchargée » peut être représenté par un bigraphe B_{φ} qui exprime cette affirmation de manière bigraphique. Les actions déclenchées par les stratégies sont les règles de réaction Ri précédemment définies.

États des entités du système Cloud

Avant d'introduire nos stratégies, nous expliquons dans un premier temps le principe d'états des entités du Cloud et comment cet état est déterminé. Lors du fonctionnement d'un système Cloud, les différentes entités le composant (serveur, VM, instance de service) peuvent changer d'état en termes de leur élasticité. Par exemple, une VM peut transiter d'un état stable vers un état où elle serait surchargée/non-utilisée, et inversement. Nous présentons ici un ensemble de règles de réaction servant à décrire l'étiquetage des différentes entités afin d'indiquer leur changement d'état. Concrètement, ces transitions sont réalisées en marquant/démarquant, lorsque cela est nécessaire, les contrôles liés aux nœuds de sorte $\widehat{evs}$ (serveurs, VMs et instances de services). Cela permet de connaitre l'état d'un nœud d'une manière apparente sans réétudier son contenu.

Par exemple, si une instance de service est considérée surchargée au-delà du seuil de 20 requêtes gérées, l'étiquetage du nœud de contrôle S la modélisant permet d'indiquer ce fait directement via le contrôle S^O. Il devient alors possible d'alléger les représentations graphiques et algébriques en abstrayant toutes les requêtes (nœuds de contrôle R) imbriquées dans un nœud S via un site di. Ainsi, grâce au marquage, les expressions $(S^O.di)$ et $(S.(R_1 \mid R_2 \mid R_3 \mid \ldots \mid R_{20}))$ deviennent équivalentes du point de vue de l'élasticité, ce qui réduit considérablement la complexité du modèle bigraphique. Le Tableau 6.4 présente algébriquement les principales règles de réaction $(R15 - R21)$ qui permettent d'étiqueter les différents nœuds de l'état stable vers un autre état (surchargé, sur-dimensionné ou non-utilisé). Ces règles servent principalement à montrer l'évolution de l'état global du système à travers les états des entités le composant. Elles peuvent ainsi être associées aux règles $R1 - R14$ précédemment définies afin de décrire le comportement élastique dynamique du système Cloud. De manière générale, cet étiquetage consiste à modifier le

contrôle d'un nœud pour en indiquer un autre état. Sémantiquement, cette modification n'altère pas l'intégrité du modèle en termes de placement, étant donné que les contrôles utilisés sont de même sorte. Ainsi, la logique de typage imposée pour le bigraphe CS reste préservée.

TABLE 6.4 – Règles de réaction pour l'étiquetage des états

Marquage	Règles de réaction
Serveur surchargé	$R15 \stackrel{\text{def}}{=} (SE.d1) \mid id \rightarrow (SE^O.d1) \mid id$
Serveur non-utilisé	$R16 \stackrel{\text{def}}{=} (SE.d1) \mid id \rightarrow (SE^I.d1) \mid id$
VM surchargée	$R17 \stackrel{\text{def}}{=} (SE.(VM.d2) \mid d1) \mid id \rightarrow (SE.(VM^O.d2) \mid d1) \mid id$
VM sur-dimensionnée	$R18 \stackrel{\text{def}}{=} (SE.(VM.d2) \mid d1) \mid id \rightarrow (SE.(VM^P.d2) \mid d1) \mid id$
VM non-utilisée	$R19 \stackrel{\text{def}}{=} (SE.(VM.d2) \mid d1) \mid id \rightarrow (SE.(VM^I.d2) \mid d1) \mid id$
Service surchargé	$R20 \stackrel{\text{def}}{=} (SE.(VM.(S.d3) \mid d2) \mid d1) \mid id \rightarrow (SE.(VM.(S^O.d3) \mid d2) \mid d1) \mid id$
Service non-utilisé	$R21 \stackrel{\text{def}}{=} (SE.(VM.(S.d3) \mid d2) \mid d1) \mid id \rightarrow (SE.(VM.(S^I.d3) \mid d2) \mid d1) \mid id$

Afin d'expliquer le principe de ces règles de réaction, prenons l'exemple de la règle $R15$. Algébriquement, elle décrit un *redex* où un nœud de contrôle SE (serveur stable) est déployé. Le *redex* se réécrit en un *reactum* où le nœud change d'état en prenant le contrôle SE^O (serveur surchargé). Sémantiquement, le changement de contrôle n'affecte pas l'intégrité du modèle en termes de placement structurel puisque les deux contrôles SE et SE^O sont de la même sorte e. La forme graphique de la règle $R15$ est donnée par la Figure 6.8.

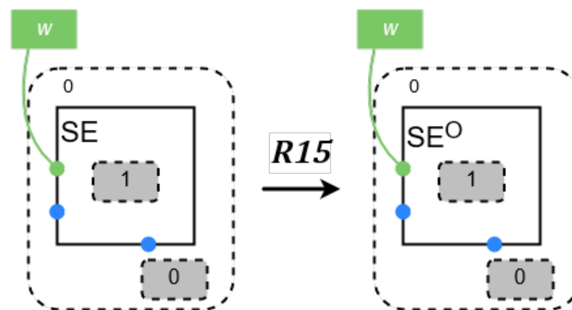


FIGURE 6.8 – Forme graphique de la règle de réaction $R15$

Déclenchement des règles de réaction pour l'étiquetage des états. Nous expliquons ici comment les règles de réaction ($R15 - R21$) permettant d'étiqueter les états des différents entités du système sont déclenchées.

Dans le méta-modèle décrivant l'architecture d'un système Cloud (voir Section 6.2), nous avions identifié quelques notions clé au niveau des entités composant le système. Il s'agit de leur charge de travail (*load*) et des seuils de capacité maximale et mini-

male (*max/min capacity*) liés à cette charge de travail. Pour un serveur (VM, instance de service), la charge de travail (*load*) donne le nombre de VMs (instances de services et requêtes) hébergées, et *max/min capacity* est utilisé pour déterminer son état. De manière générale, le *load* d'une entité est comparé à ses seuils de capacité pour en déterminer l'état (stable, surchargé, sur-dimensionné ou non utilisé). Nous introduisons les fonctions *load*, *max* et *min* pour exprimer ces valeurs. En termes de graphe de places du bigraphe CS, le *load* revient à donner le nombre de fils du nœud x passé en paramètres.

Pour toute entité x représentant un serveur, une VM ou une instance de service, les règles de réaction de marquage d'état sont déclenchées selon les cas suivants :

— *Marquage surchargé* : lorsque $load(x) \geq max(x)$, déclencher $R15$, $R17$ ou $R20$ si x est un serveur, un VM ou un service.

— *Marquage non-utilisé* : lorsque $load(x) = 0$, déclencher $R16$, $R18$ ou $R21$ si x est un serveur, une VM ou un service.

— *Marquage sur-dimensionné* : lorsque $load(x) \leq min(x)$, déclencher $R18$ lorsque x est une VM.

6.3.1 Dimensionnement Horizontal

Afin de contrôler le déclenchement des différentes règles d'élasticité horizontale ($R1-R6$), nous introduisons des stratégies décrivant le raisonnement à appliquer afin d'assurer les actions d'ajout et de retrait des ressources aux niveau infrastructure et application du système Cloud. Pour l'ajout des ressources Cloud (*scale-out*), nous définissons deux stratégies différentes : la première (H_Out1) assure une haute disponibilité des ressources, et la deuxième (H_Out2) en décrit une disponibilité limitée. Pour le retrait des ressources Cloud (*scale-in*), nous introduisons la stratégie H_In. Dans cette sous-section, nous expliquons informellement chaque stratégie en identifiant ses conditions déclenchantes et actions liées. Le Tableau 6.5 regroupe et donne la définition formelle des stratégies d'élasticité horizontale. Les conditions φi sont exprimées formellement à l'aide d'expressions de la logique du premier ordre et les actions sont données en termes de règle de réaction Ri.

Stratégie H_Out1 (Scale-Out / Haute disponibilité). Cette première stratégie d'élasticité horizontale est utilisée pour assurer une haute disponibilité des différentes ressources Cloud. Il s'agit d'autoriser l'ajout de ressources en appliquant la règle de réaction associée. Cette stratégie peut être appliquée au niveau infrastructure ou au niveau application de la manière suivante.

— *Niveau infrastructure :* à ce niveau, la stratégie concerne les serveurs physiques et les instances de machines virtuelles (VMs). Elle réagit aux conditions « il existe un serveur surchargé et aucun autre serveur n'est non-utilisé » et « il existe une VM surchargée et aucune autre n'est non-utilisée » respectivement exprimées à travers les prédicats $\varphi 1a$ et $\varphi 3a$. La vérification de l'existence consiste à vérifier s'il existe ($\exists$) un nœud, appartenant à ($\in$) l'ensemble de nœud V_{CS} du bigraphe CS, dont le contrôle ($ctrl_{CS}(noeud)$) est ($=$) SE^O ou VM^O (càd. Serveur ou VM surchargé(e)). La non existence revient à s'assurer que tous les nœuds ($\forall$) ne soient pas ($\neq$) de contrôle SE^I ou VM^I (serveur ou VM non utilisé(e)). Si le prédicat $\varphi 1a$ est vérifié, le contrôleur d'élasticité applique la règle de réaction $R1$ afin de mettre un nouveau

Table 6.5 – Stratégies d'élasticité Horizontale

Niveau	Condition	Action
H_Out1 (Scale-Out : haute disponibilité)		
Infrastructure	Un serveur est *surchargé* et aucun autre n'est *non-utilisé* $\varphi 1a \equiv \forall e \in V_{CS}\ \exists e' \in V_{CS}\ ctrl_{CS}(e') = SE^O \wedge ctrl(e) \neq SE^I$	$R1$
	Une VM est *surchargée* et aucune autre n'est *non-utilisée* $\varphi 3a \equiv \forall v \in V_{CS}\ \exists v' \in V_{CS}\ ctrl_{CS}(v') = VM^O \wedge ctrl(e) \neq VM^I$	$R3$
Application	Une instance de service est *surchargée* et aucune autre n'est *non-utilisée* $\varphi 5a \equiv \forall s \in V_{CS}\ \exists s' \in V_{CS}\ ctrl_{CS}(s') = S^O \wedge ctrl(s) \neq S^I$	$R5$
H_Out2 (Scale-Out : disponibilité limitée)		
Infrastructure	Tous les serveurs sont *surchargés* $\varphi 1b \equiv \forall e \in V_{CS}\ ctrl_{CS}(e) = SE^O$	$R1$
	Toutes les VMs sont *surchargées* $\varphi 3b \equiv \forall v \in V_{CS}\ ctrl_{CS}(v) = VM^O$	$R3$
Application	Toutes les instances de service sont *surchargées* $\varphi 5b \equiv \forall s \in V_{CS}\ ctrl_{CS}(s) = S^O$	$R5$
H_In (Scale-In)		
Infrastructure	Un serveur est *non-utilisé* et aucun autre n'est *surchargé* $\varphi 2 \equiv \forall e \in V_{CS}\ \exists e' \in V_{CS}\ ctrl_{CS}(e) \neq SE^L \wedge ctrl_{CS}(e') = SE^I$	$R2$
	Une VM est *non-utilisée* et aucune autre n'est *surchargée* $\varphi 4 \equiv \forall v \in V_{CS}\ \exists v' \in V_{CS}\ ctrl_{CS}(v) \neq VM^L \wedge ctrl_{CS}(v') = VM^I$	$R4$
Application	Une instance de service est *non-utilisée* et aucune autre n'est *surchargée* $\varphi 6 \equiv \forall s \in V_{CS} \exists s' \in V_{CS} ctrl_{CS}(s) \neq S^L \wedge ctrl_{CS}(s') = S^I$	$R6$

serveur physique en ligne. De la même manière, la règle de réaction $R3$ est appliquée, lorsque le prédicat $\varphi 3a$ est vrai, pour déployer une nouvelle VM.

— *Niveau application :* la stratégie s'intéresse à l'état d'approvisionnement du système en termes d'instances de service. Elle réagit à la condition « il existe une instance de service surchargée et aucune autre n'est non-utilisée », exprimée par le prédicat $\Phi 5a$. Lorsque la condition est vérifiée, la règle de réaction $R5$ est appliquée afin de rajouter une nouvelle instance de service.

Stratégie H_Out2 (Scale-Out / Disponibilité limitée). Cette deuxième stratégie d'élasticité horizontale est utilisée pour imposer une disponibilité limitée en termes d'ajout de ressources du Cloud (scale-out). Très similaire à la première stratégie, elle diffère au niveau des conditions déclenchantes. Cette stratégie peut être appliquée au niveaux infrastructure et application de la manière suivante.

— *Niveau infrastructure* : il s'agit de contrôler l'ajout de serveurs et VMs si les conditions « tous les serveurs sont surchargés » et « toutes les VMs sont surchargées » respectivement. Ces conditions sont exprimées par les prédicats $\varphi 1b$ et $\varphi 3b$ où la quantification universelle ($\forall$) est appliquée pour vérifier que tous les nœuds de CS

sont de contrôle SE^O et VM^O. Lorsque les prédicats $\varphi 1b$ et $\varphi 3b$ sont vérifiés, les règles $R1$ et $R3$ sont appliquées pour ajouter un nouveau serveur ou une nouvelle VM respectivement.
— *Niveau application* : une instance de service est déployée en appliquant la règle de réaction $R5$ si la condition « toutes les instances de service sont surchargées » ($\Phi 5b$) est vérifiée.

Stratégie H_In (Scale-In). Cette stratégie décrit le retrait des ressources Cloud (scale-in) en contrôlant l'application des actions d'adaptation adéquates. Elle est appliquée au niveaux infrastructure et application de la manière suivante.

— *Niveau infrastructure :* le retrait des serveurs et VMs non utilisés en appliquant les règles de réaction $R2$ et $R4$, se fait respectivement sous les conditions suivantes. « il existe un serveur non utilisé (SE^I) et aucun autre serveur n'est surchargé » ($\varphi 2$) ; et « il existe une VM non utilisée (VM^I) et aucune autre VM n'est surchargée » ($\varphi 4$).
— *Niveau application :* le retrait d'une instance de service est autorisé si la condition «il existe une instance de service non utilisée (S^I) et aucune autre instance de service n'est surchargée » ($\varphi 6$). Si la condition est vérifiée, la règle de réaction $R6$ est appliquée.

Le choix de ne pas systématiquement retirer une entité *non-utilisée* est primordial pour éviter les boucles dans l'adaptation élastique autonomique. Par exemple, une nouvelle VM est déployée si une VM est surchargée en appliquant la stratégie H_Out1 et si toutes les VMs sont surchargées en appliquant la stratégie H_Out2. Or, la nouvelle VM déployée est initialement non-utilisée et il est indésirable de la retirer. Les stratégies d'élasticité horizontale sont complémentées par les stratégies de migration et de load balancing où les VM, instances de service et requêtes sont déplacées vers les ressources récemment déployées afin de réduire la charge de leur entités hôtes. Ce choix permet de prévenir les boucles dans l'adaptation élatique où des actions contraires sont exécutées successivement, assurant ainsi la propriété de *resources thrashing* [Bersani et al., 2014].

6.3.2 Migration et Load Balancing

Afin de contrôler le déclenchement des règles de réaction pour la migration ($R11 - R12$) et le load balancing ($R13 - R14$), nous introduisons deux stratégies Mig et LB complémentaires avec les stratégies d'élasticité horizontal précédemment définies. Généralement, les actions de migration et de load balancing sont appliquées afin d'équilibrer la charge de travail des entités hôtes composant le système Cloud (serveur, VM, instance de service). Cela consiste à déplacer leur entités hébergées (VM, service, requêtes) vers des entités moins chargées que leurs hôtes afin d'équilibrer la charge globale des entités déployées. Ces stratégies viennent compléter les stratégies d'élasticité horizontale afin d'utiliser les entités nouvellement déployées et de décharger celles qui sont surchargées. Le Tableau 6.6 regroupe et donne la définition formelle des stratégies de migration et de load balancing.

Stratégie de migration (Mig). La migration est appliquée aux niveaux infrastructure et application de la manière suivante.

— *Niveau infrastructure* : La migration réagit à la condition « un serveur est surchargé et un autre serveur ne l'est pas » ($\varphi 11$). La règle $R11$ est appliquée quand cette condition est vérifiée.

— *Niveau application* : La règle $R12$ est appliquée quand la condition « une VM est surchargée et une autre VM ne l'est pas » ($\varphi 12$).

Stratégie de load balancing (LB). Le load balancing ou l'équilibrage de charge est appliqué au niveau application. Cela consiste à rediriger des requêtes d'une instance de service surchargée à une autre qui ne l'est pas. La stratégie LB réagit à la condition « une instance de service est *surchargée* et une autre ne l'est pas » dans le cas où l'instance non chargée se trouve dans une VM distante ($\varphi 13$) ou bien dans la même VM hôte ($\varphi 14$). Les règles de réaction $R13$ et $R14$ sont déclenchées quand ces conditions sont vérifiées.

Notons qu'au niveau des prédicats $\varphi 11 - \varphi 14$, nous incluons la condition $(load(x) - load(x') > 1)$ où x et x' sont des nœuds de même sorte avec x de contrôle surchargée et x' de contrôle non surchargé (stable ou non-utilisé). Cette condition sert à éviter de surcharger une entité stable en déchargeant une entité surchargée. En effet, cela créerait une boucle où les nœuds x et x' serait simplement interchangés. Il faudrait donc que la différence de charge (*load*) soit supérieure à 1. Aussi, pour identifier la VM hôte de chaque nœud, la fonction de parentalité $prnt_{CS}$ du bigraphe CS est utilisée afin d'identifier la même VM hôte ou une VM distante.

Table 6.6 – Stratégies de migration et de load balancing

Niveau	Condition	Action
Mig (migration)		
Infrastructure	Un serveur est surchargé et un autre serveur ne l'est pas $\varphi 11 \equiv \exists e, e' \in V_{CS}\ ctrl(e) = SE^O \wedge ctrl(e') \neq SE^O \wedge load(e) - load(e') > 1$	$R11$
Application	Une VM est surchargée et une autre VM ne l'est pas $\varphi 12 \equiv \exists v, v' \in V_{CS}\ ctrl(v) = VM^O \wedge ctrl(v') \neq VM^O \wedge load(v) - load(v') > 1$	$R12$
LB (load balancing)		
Application	Un service est surchargé et un autre d'une VM distante ne l'est pas $\varphi 13 \equiv \exists s, s' \in V_{CS}\ ctrl(s) = S^O \wedge ctrl(s) \neq S^O \wedge load(s) - load(s') > 1 \wedge prnt(s) \neq prnt(s')$	$R13$
	Un service est surchargé et un autre de la même VM ne l'est pas $\varphi 14 \equiv \exists s, s' \in V_{CS}\ ctrl(s) = S^O \wedge ctrl(s') \neq S^O \wedge load(s) - load(s') > 1 \wedge prnt_{CS}(s) = prnt_{CS}(s')$	$R14$

6.3.3 Dimensionnement Vertical

Le dimensionnement vertical consiste à augmenter ou diminuer l'offre (*offering*) d'une VM en termes de ressources informatiques (unités de CPU et de RAM). La modification de l'offre d'une VM dépend de l'offre de son serveur physique hôte, en termes de nombre de ressources disponibles. Nous introduisons deux fonctions $cpu(x)$ et $ram(x)$ qui renvoient la quantité d'unités de CPU et de mémoire RAM disponibles au niveau de l'entité x en paramètres, pouvant être un serveur physique ou une machine virtuelle. D'un point de vue du graphe des places du bigraphe CS, les fonctions cpu/ram renvoient le nombre de

nœuds de contrôle CU/M hébergés par le nœud de sorte r (ressources), lui-même hébergé par le nœud de sorte v/e (VM ou serveur) passé en paramètres.

Nous définissons deux stratégies pour le dimensionnement verticale. La première est consacrée à l'ajout des ressources (*scale-up*) et la deuxième à leur retrait (*scale-down*). Le Tableau 6.7 regroupe et exprime formellement ces deux stratégies.

Table 6.7 – Stratégies de dimensionnement Vertical

Ressource	Condition	Action
	V_UP (Scale-Up)	
CPU	Une VM est surchargée et il y a plus de CPU que de RAM disponibles $\varphi7 \equiv \exists v, e \in V_{CS}\ ctrl(v) = VM^O \wedge prnt(v) = e \wedge cpu(e) \geq ram(e) > 0$	$R7$
RAM	Une VM est surchargée et il y a plus de RAM que de CPU disponibles $\varphi9 \equiv \exists v, e \in V_{CS}\ ctrl(v) = VM^O \wedge prnt(v) = e \wedge ram(e) \geq cpu(e) > 0$	$R9$
	V_Down (Scale-Down)	
CPU	Une VM est sur-dimensionnée et utilise plus de CPU que de RAM $\varphi8 \equiv \exists v \in V_{CS}\ ctrl(v) = VM^P \wedge cpu(v) \geq ram(v) \wedge cpu(v) > 1$	$R8$
RAM	Une VM est sur-dimensionnée et utilise plus de RAM que de CPU $\varphi10 \equiv \exists v \in V_{CS}\ ctrl(v) = VM^P \wedge ram(v) \geq cpu(v) \wedge ram(v) > 1$	$R10$

Stratégie V_UP (Scale-Up). Lorsqu'une machine virtuelle est surchargée (VM^O), ou lorsque sa charge de travail est supérieure à son seuil de capacité maximale, il est nécessaire d'un point de vue dimensionnement vertical de lui allouer plus de ressources (*scale-up*). Cependant, la quantité à allouer dépend de la disponibilité des ressources au niveau du serveur physique hôte. Ainsi, lorsque ≪ la quantité d'unités CPU disponibles est suffisante et supérieure ou égale à la quantité d'unités de mémoire RAM ≫ ($\varphi7$), une unité de CPU en plus en allouée à la VM ($R7$). Réciproquement, lorsque ≪ la quantité de mémoire RAM disponible est positive et supérieure ou égale à la quantité de CPU ≫ ($\varphi9$), une unité de RAM supplémentaire est allouée à la VM ($R9$).

Stratégie V_Down (Scale-Down). Quand la machine virtuelle est sur-provisionnée (VM^P), ou lorsque sa charge de travail est inférieure à son seuil de capacité minimale, on dit que cette VM est en situation de ≪ surplus ≫ et doit libérer les ressources allouées en trop (*scale-down*). Afin d'éviter un déséquilibre entre la quantité de RAM et de CPU, La stratégie V_Down raisonne de la manière suivante. Lorsque ≪ la quantité d'unités CPU allouée est supérieure ou égale à la quantité d'unités de mémoire RAM ≫ ($\varphi8$), une unité de CPU est libérée de la VM et rendue au serveur hôte (R8). Réciproquement, lorsque ≪ la quantité de mémoire RAM allouée est supérieure ou égale à la quantité de CPU ≫ ($\varphi10$), une unité de RAM est libérée de la VM et rendue au serveur hôte ($R10$). Les deux conditions s'assurent qu'il restera au moins une unité de chaque ressource à la VM nécessaires pour son fonctionnement.

6.3.4 Bilan

À ce stade de nos contributions, nous avons spécifié formellement l'architecture des systèmes Cloud ainsi que leur comportement élastique désirés. Nous avons défini une sémantique bigraphique accompagnée de sa discipline de typage pour modéliser les aspects structurels d'un système Cloud. Ensuite, nous avons modélisé par le biais d'une sémantique basée sur les BRS, les différentes actions d'adaptation pour l'élasticité horizontale et verticale, ainsi que les actions de migration et de load balancing aux niveaux infrastructure et application d'un système Cloud. Enfin, nous avons introduit des stratégies d'élasticité décrivant le comportement désiré du contrôleur d'élasticité pour la gestion dynamique de l'élasticité. Ainsi, nous avons atteint notre premier objectif (OR1) visant à spécifier formellement une structure et des comportements élastiques d'un système Cloud (voir Section 5.1).

6.4 Conclusion

Dans ce chapitre, nous avons présenté une approche formelle à base des bigraphes (accompagnés de leur discipline de typage) et des systèmes réactifs bigraphiques (BRS) pour la spécification des aspects structurels et comportementaux des systèmes Cloud élastiques respectivement.

Dans un premier temps, nous avons proposé une approche générique à base des bigraphes pour exprimer les éléments pertinents d'une architecture Cloud en multi-couche, c'est à dire aux niveaux infrastructure et application. La logique de typage associée aux bigraphes conçus permet de décrire une sémantique robuste et détaillées pour les systèmes Cloud et les entités les composant (serveurs physiques, machines virtuelles, instances de service Cloud, requêtes, ressources informatiques déployées).

Ensuite, Nous avons proposé une sémantique basée sur les BRS pour la modélisation des actions de reconfiguration dynamiques des systèmes Cloud, en termes de règles de réaction bigraphiques, s'appliquant au niveau des différentes ressources matérielles et logicielles déployées au sein du système (multi-couches). À partir des différentes actions identifiées, nous avons défini plusieurs stratégies d'élasticité afin de décrire une logique pour le contrôle des redimensionnements du système. Précisément, nous avons introduit des stratégies décrivant les différentes méthodes d'élasticité (horizontale, verticale, migration et load balancing). Ces stratégies peuvent s'appliquer en multi-couche, c'est-à-dire, à différents niveaux du système (infrastructure, application, ressources). Elles décrivent comment les actions de reconfigurations (règles de réaction bigraphiques) peuvent être déclenchées afin de permettre au système Cloud de s'adapter à sa charge de travail de manière réactive, tout en préservant sa cohérence architecturale.

Dans le chapitre suivant, nous montrons comment les spécifications bigraphiques ainsi que les stratégies d'élasticité spécifiées peuvent être encodées dans le langage de spécification formelle Maude. Nous montrons la complémentarité de Maude et des BRS pour l'exécution et la vérification du bon fonctionnement des comportements régis par le contrôleur d'élasticité.

Chapitre 7

Implémentation et vérification du comportement élastique d'un système Cloud

Sommaire

7.1 Introduction 104
7.2 Encodage des spécifications bigraphiques dans Maude 105
7.2.1 Principes de l'encodage 105
7.2.2 Encodage des aspects structurels (module fonctionnel) 106
7.2.3 Encodage des stratégies d'élasticité (module système) 108
7.2.4 Bilan 110
7.3 Vérification du comportement élastique 111
7.3.1 Comportement et propriétés élastiques 111
7.3.2 Encodage d'états et de propriétés dans Maude 114
7.3.3 Vérification de propriétés 115
7.3.4 Bilan 118
7.4 Conclusion 118

7.1 Introduction

Les systèmes réactifs bigraphiques incarnent un excellent moyen de modéliser les aspects structurels et comportementaux des systèmes Cloud élastiques. Cependant, les outils développés autour des BRS tels que BPL Tool [Højsgaard and Glenstrup, 2011], BigMC [Perrone et al., 2012] et BigraphER [Sevegnani and Calder, 2016] ne répondent pas à nos exigences en termes d'expressivité et de performance. Ces outils ne sont pas encore matures et représentent des prototypes ayant plusieurs limites. Nous retiendrons deux limites principales rencontrées dans nos travaux. Au meilleur de nos connaissances, les outils existants ne permettent pas de : (1) définir des règles de réaction à déclenchement conditionnel, où les conditions seraient exprimées dans un langage (e.g. logique du premier ordre) permettant la quantification universelle et existentielle des entités dans le contexte global du système. Ces outils ne permettent pas non plus de (2) raisonner sur l'état global du système d'un point de vue quantitatif. Par exemple, définir pour un nœud un attribut indiquant son nombre maximum de fils, avoir une mécanique permettant de savoir quand ce seuil est atteint et enfin connaitre le nombre de nœuds dont ce seuil aurait été atteint. Logiquement, de ces limitations pratiques découle l'incapacité des outils

existants à exprimer nos stratégies d'élasticité et donc à fournir une vérification formelle complète des comportements qu'elles décrivent. À titre d'exemple, l'outil BigMC, unique model-checker destiné aux BRS, est très limité en ce qui concerne l'expression des propriétés complexes et ne supporte pas la logique de typage des BRS. En outre, l'outil BigraphER fournit un cadre pour manipuler, visualiser et simuler des BRS. Cependant, il n'est pas équipé de model-checker et ne permet donc pas de vérifier formellement les modèles bigraphiques conçus.

Les différentes limites citées nous ont orienté vers la recherche d'un langage permettant de fournir des mécanismes de vérification formelle suffisants. Cette recherche nous a mené à l'utilisation du langage Maude pour encoder les BRS spécifiés en les étendant des notions (quantitative et conditionnelle) voulues. Il permet également d'exécuter les comportements spécifiés, notamment en permettant l'implémentation des différentes stratégies d'élasticité définies. Enfin, Maude dispose d'un model-checker qui repose sur la logique temporelle linéaire LTL. Il permet de vérifier formellement le comportement complexe des systèmes Cloud élastiques et plus particulièrement celui du contrôleur d'élasticité.

Dans ce Chapitre, nous proposons une solution pour l'implémentation, l'exécution et la vérification qualitative des stratégies d'élasticité introduites. Précisément, nous recourrons au langage de spécification formelle Maude qui offre un cadre sémantique à base de théorie de réécriture. Dans la Section 7.2, nous détaillons la complémentarité de Maude et des BRS en fournissant un encodage, dans le langage Maude, des spécification bigraphiques pour les structures du Cloud et de leur comportement élastique. Nous y présentons l'une des principales originalités de notre approche, à savoir la description des états d'un système Cloud du point de vue de l'élasticité, en analysant ses configurations multi-couches par le biais de prédicats de la logique de premier ordre. Dans la Section 7.3, nous présentons une approche de vérification formelle des comportements élastiques introduits. Nous y proposons une solution de vérification formelle via une technique de model-checking à base d'états, supportée par la logique temporelle linéaire (LTL).

7.2 Encodage des spécifications bigraphiques dans Maude

Dans cette Section, nous proposons un moyen d'encoder, dans la langage Maude, les spécifications bigraphiques ainsi que le comportement du contrôleur d'élasticité à travers des différentes stratégies d'élasticité définies.

7.2.1 Principes de l'encodage

D'un point de vue mathématique, les modules fonctionnels et systèmes de Maude (voir Section 4.3) sont construits de la manière suivante [Clavel et al., 2016] :

1. Le module fonctionnel spécifie une théorie en logique équationnelle d'appartenance. Une telle théorie est un couple $(\Sigma, E \cup A)$, où la signature Σ spécifie la structure de type (sortes, sous-types, opérateurs, etc.), E est l'ensemble des équations éventuellement conditionnelles déclarées dans le module fonctionnel, et A est l'ensemble des attributs d'équation déclarés pour les opérateurs (associativité, commutativité, etc.).
2. Le module système qui spécifie une théorie de réécriture sous forme de triple $(\Sigma, E \cup A, R)$. Où $(\Sigma, E \cup A)$ est la partie théorie équationnelle du module, et R est un ensemble de règles de réécriture potentiellement conditionnelles.

Ainsi, (1) les aspects structurels des spécifications bigraphiques (signature et logique de typage), sont encodées dans le module fonctionnel *Elastic_Cloud_System*. Où les opérations et équations déclarées définissent des constructeurs et des axiomes qui construisent les éléments du système et capturent leurs états. De même, (2) la dynamique introduite par les BRS (règles de réaction) est traduite dans un module système *Elastic_Cloud_Behavior* définissant un ensemble règles de réécriture R conditionnelles. Concrètement, la spécification dans le langage Maude permet d'encoder les spécifications bigraphiques de manière à préserver leur sémantique d'une part tout en permettant de les enrichir d'une autre. Notamment en encodant des informations quantitatives sur les entités du système et en spécifiant des règles de réécriture conditionnelles de manière à intégrer la quantification universelle et existentielle sur ces entités. Cela permettrait d'implémenter les différentes stratégies d'élasticité proposées et donc de contrôler le comportement complexe d'un système Cloud élastique en raisonnant sur son état global. La Figure 7.1 donne une vision d'ensemble du principe d'encodage dans Maude des spécifications bigraphiques ainsi que le comportement autonomique du contrôleur d'élasticité.

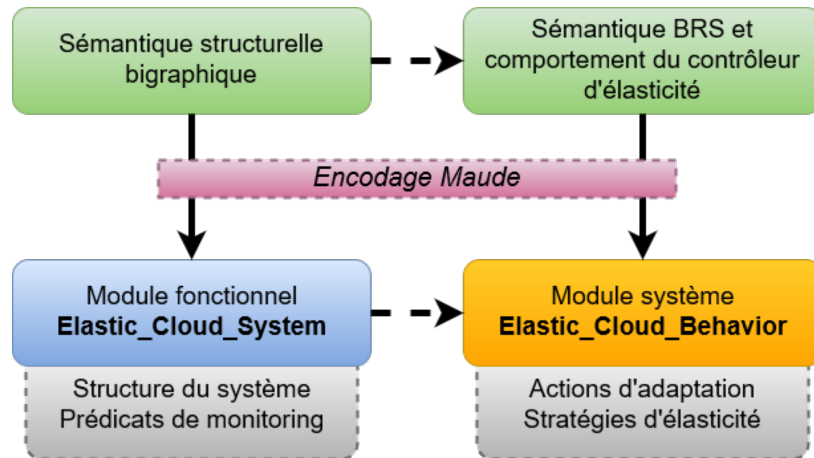


FIGURE 7.1 – Vue d'ensemble du principe d'encodage des spécifications bigraphiques dans Maude

7.2.2 Encodage des aspects structurels (module fonctionnel)

La sémantique bigraphique pour les systèmes Cloud est traduite dans un module fonctionnel *Elastic_Cloud_System*. Ce module intègre la sémantique de typage (sortes), les relations de sous-sortes et les opérations algébriques utilisée pour définir la structure syntaxique des différents éléments d'un système Cloud. En vue de la complexité des comportements élastiques définis, nous présentons ici un encodage considérant un système Cloud sur un seul serveur physique. Le module fonctionnel *Elastic_Cloud_System* est construit en deux parties. La première définit structurellement les entités du système et la deuxième donne les différentes opérations et prédicats servant à capturer son état. Le Tableau 7.1 regroupe les déclarations les plus pertinentes définies dans ce module.

Structure du système. Les sortes précédemment définies (e : serveur, v : VM, s : service, r : ressources) sont préservées et exprimées par les sortes CS, VM, S et Ressources respectivement. L'encodage dans Maude permet également d'enrichir la spécification des entités du Cloud en leur affectant un seuil d'hébergement maximal max et un état state. Syntaxiquement, une sorte est déclarée via le mot-clé `sort` et est définie par un

Table 7.1 – Principales déclarations du module fonctionnel *Elastic_Cloud_System*

Module fonctionnel *Elastic_Cloud_System*

Sortes et structure du système

```
sorts CS VM S Resources VML SL state.
subsort VM < VML . subsort S < SL .
op CS<_ ,_ ,_ /_ :_ :_ > : Nat Nat Nat VML Resources state -> CS [ctor] .
op VM_ ,_ :_ :_  : Nat SL Resources state -> VM [ctor] .
op S[_ ,_ :_ ] : Nat Nat state -> S [ctor] .
op [_ & _ ] : Nat Nat -> Resources [ctor] .
ops stable over idle ... : -> state [ctor] .
...
```

Opérations et prédicats

```
op loadCS(_): CS -> Nat .
op loadVM(_): VM -> Nat .
op loadS(_) : S -> Nat .
op getResourcesCS(_) : CS -> Resources .
op getCpu(_) getRam(_) : CS -> Nat .
op getResourcesV(_) : VM -> Resources .
ops getVcpu(_) getVram(_) : VM -> Nat .
ops isStable(_) isOverloaded(_) isUnderused(_) AoverV(_) EoverV(_)
EunV(_) AoverS(_) EoverS(_) EunS(_) ... : CS -> Bool .
ops stableV(_) overV(_) idleV(_) ... : VM -> Bool .
ops stableS(_) overS(_) idleS(_) ... : S -> Bool .
ops lessS(_) mostS(_) : CS -> S .
ops lessV(_) mostV(_) : CS -> VM .
...
```

constructeur (ctor) associé. Par exemple, Le système Cloud est construit par le terme `CS<x,y,z/VML:Resources:state>`, où `x`, `y` et `z` sont des entiers naturels représentant les seuils max initiaux pour le serveur, les machines virtuelles et les instances de service. `VML` est une sorte représentant l'ensemble des VM déployées au sein du serveur. La sorte VM est donc déclarée comme une sous-sorte de `VML`, à travers le mot-clé `subsort`. Le sorte `state` attribue un état au serveur parmi différents constructeurs (e.g. `overloaded`, `idle`, `stable`, etc.). Enfin, `Resources` est une sorte qui donne la valeur des ressources disponibles au niveau du serveur. Elle est donnée par le constructeur [`cpu` & `ram`] où cpu et ram sont exprimés en termes d'unités à travers des entiers naturels. De la même manière, les sortes VM et S sont définies à travers leur constructeur respectifs. Par exemple, le constructeur d'une instance de service est donné par : `S[z,load:state]`. Où z est sa capacité d'hébergement *max*, `state` est son état et `load` est un entier nature donnant le nombre de requêtes prises en charges.

Prédicats et états du système. Afin de capturer les différents états du système, nous définissons un ensemble d'opérations et prédicats. Ce sont des opérations déclarées par le mot-clé op qui prennent en paramètres différentes structures telles que CS, VM, S, etc. et qui renvoient des informations les concernant comme leur charge de travail (*load*), la quantité de ressources qu'elles abritent ou une valeur de vérité concernant la nature

du prédicat. Ces opérations et prédicats servent pour le monitoring du système et la surveillance des différentes situations pouvant s'y produire. Par exemple, l'opération *AoverV* est un prédicat exprimant ≪ toutes les VM sont surchargée ≫ et *EunS* est un prédicat pour ≪ il existe une instance de service non-utilisée ≫. D'une part, ces prédicats serviront à encoder les différents prédicats φ_i gouvernant le déclenchement des stratégies d'élasticité. D'autre part, ils serviront à affiner et améliorer les actions d'élasticité appliquées. Par exemple, nous étendons l'expressivité du modèle en introduisant des opérations telles que *lessS/V* ou *mostS/V* afin de détecter l'instance de service/VM la moins ou plus chargée en termes de capacité d'hébergement. Cela servirait à mieux choisir les VMs et instances de service cible pour le load balancing et la migration. Notons que la déclaration d'opérations différentes ayant la même signature peuvent être regroupée par le mot-clé `ops`.

7.2.3 Encodage des stratégies d'élasticité (module système)

Les différentes règles de réaction exprimant les opérations de dimensionnement élastique, ainsi que les différentes stratégies d'élasticité sont encodées dans un module système Maude nommé *Elastic_Cloud_Behavior*. Les règles de réécriture encodant les actions de dimensionnement horizontal et vertical, le load balancing et la migration $R1 - R14$ (voir Section 6.2) sont exprimés par des fonctions de réécriture Maude. Les règles de réaction pour l'étiquetage des états ($R15 - R21$) ainsi que les stratégies d'élasticité sont exprimées par des règles de réécriture conditionnelles. Le Tableau 7.2 regroupe les définitions les plus pertinentes de ce module. À l'instar des règles de réaction bigraphiques, les règles de réécriture conditionnelles de Maude consistent à réécrire la partie gauche de la règle en sa partie droite si les conditions de déclenchement spécifiées sont vérifiées. La partie gauche de la règle décrit une configuration du système et sa partie droite exprime une reconfiguration désirée. Les conditions de déclenchement sont exprimées via les différents prédicats définis dans les deux modules fonctionnel et système ou bien par d'autres conditions simples.

Étiquetage des états. Pour les actions d'étiquetage des états, la réécriture consiste à modifier l'état des différentes entités du système si leur état évolue durant l'exécution du système. Par exemple, la règle de réécriture nommée [`mark-over-S`] indique qu'une instance de service est surchargée si son état courant ne l'indique pas déjà. Cette règle est définie de la manière suivante :

```
crl [mark-over-S] : S[st,l : sst] => S[st, l : over]
 if (sst =/= over) and (l > st) .
```

Le mot-clé `crl` pour ≪ *conditional rewrite rule*≫ déclare une règle de réécriture conditionnelle. La partie gauche est réécrite en la partie à droite du symbole (`=>`). Enfin, la condition `if (sst =/= over) and (l > st)` spécifie que l'état de l'instance de service (`sst` : service state) est différent ($\neq$) de over (surchargé) et que sa charge de travail (`l` : load) est supérieure à son seuil max (`st` : service *max threshold*).

Stratégies d'élasticité. Dans le cas des règles de réécriture encodant les stratégies d'élasticité, la réécriture dans Maude consiste à appliquer des fonctions de réductions implémentant les actions désirées. Ces fonctions sont déclarées dans le module système. Par exemple, la stratégie de migration au niveau application [`migration-S-level`] applique une réduction

fonctionnelle sur la structure de système de manière à le réécrire en appliquant l'action de migration désirée. Cette règle est spécifiée de la manière suivante :

```
crl [migration-S-level] : cs => MigS(cs) if MigSpred(cs) .
```

TABLE 7.2 – Principales déclarations du module système *Elastic_Cloud_Behavior*

Module système *Elastic_Cloud_Behavior*

Sortes et prédicats

```
sorts VSCALE HSCALE.
op VSCALE :: _ : CS -> VSCALE [ctor].
op HSCALE (V_ , S_) :: _ : Nat Nat CS -> HSCALE [ctor].
ops LBpred(_) MigSpred(_) scaleUpPredCPU(_) scaleUpPredRAM(_)
scaleDownPredCPU(_) scaleDownPredRAM(_): CS -> Bool.
...
```

Fonctions

```
ops addCpu(_) addRam(_) subCpu(_) subRam(_) HoutV1(_) HoutV2(_)
HoutS1(_) HoutS2(_) HinV(_) HinS(_) MigS(_) LB(_): CS -> CS.
...
```

Règles de réécriture

```
crl [nom-règle] : terme => terme' if condition(s)
```

```
Marquage des états
crl [mark-over-S] : S[st,l : sst]
=> S[st, l : over] if (sst =/= over) and (l > st).
crl [mar-stable-CS] : CS<ct, vt, st / VML : res : cst >
=> CS<cst,vt,st/VML : res : stable>
if (cst =/= stable) and isStable( CS<ct, vt, st / VML : res : cst >).
...
```

```
Stratégies d'élasticité
crl [migration-S-level] : cs => MigS(cs) if MigSpred(cs) .
crl [load-balancing] : cs => LB(cs) if LBpred(cs) .
crl [V-up-CPU] : VSCALE :: cs => VSCALE :: addCpu( cs ) if
scaleUpPredCPU(cs) .
crl [V-down-RAM] : VSCALE :: cs => VSCALE :: subRam( cs ) if
scaleDownPredRAM(cs) .
crl [H_Out-S1] : HSCALE (V i , S j) :: cs => HSCALE (V i , S j) ::
HoutS1( cs ) if (j == 1 and EoverS(cs) and (not EunS(cs))) .
crl [H_Out-V2] : HSCALE (V i , S j) :: cs => HSCALE (V i , S j) ::
HoutS1( cs ) if (i == 2 and AoverV(cs)) .
crl [scale-in-V] : HSCALE (V i , S j):: CS< ct,vt,st/v | vl :res:
cst> => HSCALE (V i , S j):: Vin(CS< ct,vt,st / v | vl :res: cst
>) if ((not EoverV(vl)) and unV(v)).
...
```

Le système Cloud `cs` est réécrit en appliquant dessus la fonction `MigS(cs)`, si la condition exprimée par le prédicat `MigSpred(cs)` est vérifiée. La fonction *MigS* déplace

une instance de service de la VM la plus chargée vers la VM la moins chargée et le prédicat *MigSpred* s'assure que la migration au niveau application est applicable (en implémentant la prédicat $\varphi 12$ de la Section 6.3.2).

Nous introduisons les deux sortes *VSCALE* et *HSCALE* afin de spécifier les structures supportant les stratégies d'élasticité verticale et horizontale respectivement. En effet, les règles de réécriture de Maude sont conçues pour spécifier la réécriture d'un terme en un autre de la même sorte de manière concurrente. Par conséquent, les sortes introduites serviront de socle pour restreindre le dimensionnement vertical ou horizontal d'un système, de la manière suivante.

— *Dimensionnement vertical* : La sorte VSCALE est spécifiée via le constructeur `VSCALE :: cs` où `cs` est le système Cloud géré. De cette manière, les règles de réécritures pour l'élasticité verticale sont appliquées sur un terme de cette sorte uniquement principalement afin d'éviter qu'une action de dimensionnement horizontal ne soit appliquée. Par exemple, la stratégie verticale *V_UP* d'ajout de ressources (scale-up) au niveau CPU est spécifiée par :

```
crl [V-up-CPU] : VSCALE :: cs => VSCALE :: addCpu( cs )
 if scaleUpPredCPU(cs) .
```

De cette manière, le dimensionnement vertical pour l'ajour de CPU sera assuré de manière continue en appliquant la fonction *addCpu* sur le système *cs* si le prédicat *scaleUpPredCPU(cs)* ($\varphi 7$) est vrai.

— *Dimensionnement horizontal* : La sorte HSCALE est définie par le constructeur `HSCALE (V i, S j):: cs` où les paramètres $i, j \in [1, 2]$ indiquent quelle stratégie d'ajout de ressources (*H_Out1* ou *H_Out2*) est appliquée au niveau infrastructure et application du système Cloud `cs`. Par exemple, la stratégie H_Out1 au niveau application est spécifiée par :

```
crl [H_Out-S1] : HSCALE (V i , S j) :: cs => HSCALE (V i , S j) ::
 HoutS1( cs ) if (j == 1 and EoverS(cs) and (not EunS(cs))) .
```

Lorsque la valeur du paramètre `j` indiquant la stratégie à appliquer au niveau application est égal à 1 (H_Out1), la fonction d'ajout d'une instance suivant cette stratégie est appliquée sur le système par `S1(cs)`. Le prédicat $\varphi 5$ déclenchant cette stratégie est également vérifié. Il est encodé comme suit : ≪ il existe une instance service surchargée dans le système cs (`EoverS(cs)`) et (`and`) il n'y a pas d'instance non utilisée (`not EunS(cs)`) ≫.

7.2.4 Bilan

Jusqu'à présent nous avons défini les modules fonctionnels et systèmes nécessaires pour représenter la structure des systèmes Clous élastiques ainsi que leur comportement dynamique. Cette spécification formelle dans le langage Maude obtenue peut maintenant faire l'objet d'une exécution. En effet, en utilisant l'environnement Maude qui est très versatile en matière de simulations, nous pouvons assurer le bon fonctionnement d'une partie du système ou le système complet via l'exécution des règles de réécriture. L'environnement pend en entrée une configuration initiale quelconque qui représente une architecture cloud initiale (avant son évolution ou réadaptation) et procède à son exécution en appliquant des règles de réécriture afin d'obtenir un état final (anticipé ou non). Cette simulation permet de vérifier le bon fonctionnement de chacune des règles de réécriture du mo-

dule *Elastic_Cloud_Behaviour* appliquée de manière indépendante, ou bien l'ensemble des règles de réécriture mises en commun pour modéliser un comportement autonomique, désiré ou non. À ce stade, nous avons donc répondu à notre deuxième objectif (OR2) visant à permettre l'exécution des comportements élastique désirés (voir Section 5.1). Dans la Section 7.3, nous montrons comment effectuer la vérification formelle du bon fonctionnement de ces comportements, à l'aide du model-checker de Maude.

7.3 Vérification du comportement élastique

La vérification formelle incarne une méthode très efficace pour assurer l'absence d'erreurs dans un système donné. La vérification formelle de modèles ou le "model checking" [Chechik and Gannon, 2001, Baier and Katoen, 2008] consiste à analyser d'une manière automatique un modèle qui représente une abstraction d'un système pour déterminer si une série de propriétés est satisfaite par ce modèle du système. Les propriétés représentent l'expression d'exigences envers un système, elles sont définies généralement sous la forme de formules de logique temporelle. Plus précisément, la technique de model-checking consiste à faire une recherche exhaustive et automatique au sein de l'ensemble des états possibles du système afin de vérifier s'ils répondent à des propriétés (désirables ou indésirables) exprimées en logique temporelle ou fournir un contre-exemple montrant le chemin qui a conduit à la violation d'une ou plusieurs propriétés. On parle alors de model-checking à base d'états (*state based model-checking*) [Souri et al., 2018] qui consiste à vérifier la satisfaction de certaines propriétés au niveau des états du système accessibles à partir d'un état initial donné. Dans cette Section, nous proposons une approche de vérification formelle des propriétés d'élasticité dans le Cloud selon les stratégies d'élasticité verticale et horizontale ainsi que la migration et le load balancing. Nous nous basons sur le model-checker de Maude qui repose sur la logique temporelle linéaire LTL. Pour atteindre ces objectifs, nous expliquons ici les trois étapes à suivre.

7.3.1 Comportement et propriétés élastiques

Dans cette première étape, nous définissons une structure de *Kripke* $\mathbf{A}_{CS}$ qui représente un modèle de logique temporelle pour les comportements élastiques des systèmes Cloud [Clavel et al., 2016]. Une telle structure sert à définir les différents états du système à travers des propriétés atomiques AP_{CS} en plus de définir une mécanique permettant de vérifier la satisfaction d'une propriété au niveau d'un état quelconque. Dans un premier temps, nous donnons la définition formelle de la structure de *Kripke* A_{CS}, sa configuration et montrons comment les comportements définis par une telle structure peuvent être exprimés par des systèmes de transition labélisés. Dans un deuxième temps, nous introduirons un ensemble de formules propositionnelle dans la logique temporelle linéaire $LTL(AP_{CS})$ considéré par la structure de *Kripke* et expliquerons leur rôle.

Définition de la structure de Kripke. Étant donné un ensemble AP_{CS} de propositions élémentaires, une structure de Kripke est définie par $\mathbf{A}_{CS} = (A, \rightarrow_A, L_{CS})$. Où A est l'ensemble des états du système, $\rightarrow_A$ est une relation de transition, et $L_{CS} : A \rightarrow AP_{CS}$est une fonction d'étiquetage associant à chaque état $a \in A$, un ensemble $L_{CS}(a)$ de propositions élémentaires dans AP_{CS} qui sont satisfaites à l'état a. $LTL(AP_{CS})$ définit les formules de la logique temporelle linéaire propositionnelle. La sémantique de $LTL(AP_{CS})$ est définie par la relation de satisfaction : $A_{CS}, a \vDash \Phi$, où $\Phi \in LTL(AP_{CS})$.

Configuration de la structure de Kripke. Nous considérons l'ensemble des propositions atomiques :

$$AP_{CS} = \{\varphi 1a, \varphi 1b, \varphi 2, \varphi 3a, \varphi 3b, \varphi 3, \varphi 4, \varphi 5a, \varphi 5b, \varphi 6, \varphi 7, \varphi 8, \varphi 9, \varphi 10, \varphi 11, \varphi 12, \varphi 13, \varphi 14\}.$$

Ces propositions sont indicatives des prédicats de déclenchement des différentes stratégies d'élasticité définies (voir Section 6.3). Une proposition φi est satisfaite lorsque le prédicat φi correspondant est vrai. Notons que l'état d'un système Cloud peut être très complexe étant donné qu'il dépend de plusieurs facteurs comme sa charge de travail, le nombre de ressources déployées ou encore les ressources disponibles. En fonction de ses configurations et de son contexte, un système Cloud peut donc évoluer de plusieurs manières selon les différentes stratégies d'élasticité le contrôlant. De ce fait, l'ensemble des états structurels du système (càd. Les configurations possibles d'un système Cloud *cs*) est théoriquement infini, ce qui rend la vérification des comportements du système très complexe. Pour cette raison, nous considérons les états symboliques : *Stable*, *Overloaded* (surchargé), *Overprovisioned* (sur-dimensionné) et *Unbalanced* (déséquilibré) respectivement exprimés dans l'ensemble des états considérés $A = \{S, O, P, B\}$. Ces états symboliques expriment des classes d'équivalence en vertu de l'état élastique global du système Cloud géré. En effet, plusieurs configurations structurelles différentes peuvent avoir le même état par rapport à leur élasticité. Ainsi, plusieurs configurations structurelles possibles peuvent être regroupées dans la même classe d'équivalence décrivant leur état élastique commun. Cela est accompli par le biais de la fonction d'étiquetage L_{CS} de la manière suivante :

Pour tout état $a \in A$

— Le système est à l'état *Stable* : $a = S \Leftrightarrow L_{CS}(a) = \emptyset$.

— Le système est à l'état *Overloaded* :$a = O \Leftrightarrow L_{CS}(a) \subseteq \{\varphi 1a, \varphi 1b, \varphi 3a, \varphi 3b, \varphi 5a, \varphi 5b, \varphi 7, \varphi 9\}$.

— Le système est à l'état *Overprovisioned* : $a = P \Leftrightarrow L_{CS}(a) \subseteq \{\varphi 2, \varphi 4, \varphi 6, \varphi 8, \varphi 10\}$.

— Le système est à l'état *Unbalanced* : $a = B \Leftrightarrow L_{CS}(a) \subseteq \{\varphi 11, \varphi 12, \varphi 13, \varphi 14\}$.

En d'autres termes, quand le système est à l'état surchargé (*O : Overloaded*) les actions d'élasticité horizontale ou verticale pour l'ajout des ressources (*scale-out, scale-up*) sont nécessaires. Quand il est à l'état sur-dimensionné (*P : Overprovisioned*), les actions d'élasticité horizontale ou verticale pour le retrait de ressources (*scale-in, scale-down*) sont demandées. Quand le système est à l'état déséquilibré (*B : Unbalanced*), les actions de migration et/ou de load balancing sont applicables. Enfin, quand le système est à l'état stable (S), aucune action d'adaptation élastique n'est nécessaire.

Représentation des transitions du système. Nous représentons les transitions du système à travers une notation basée sur les systèmes de transition labélisés (LTS). Les transitions décrites par la structure de Kripke permettent d'identifier les transitions possibles entre les différents états du système [Schoren, 2011]. Les états sont représentés par les états élastique symboliques introduits. Les transitions représentent les différentes actions d'adaptation précédemment définies pour l'élasticité horizontale (R1-R6), l'élasticité verticale ($R7 - R10$), la migration ($R11 - R12$) et le load balancing ($R13 - R14$). Nous représentons également les évènements in et out par des transitions. Ces évènements indiquent qu'une requête arrive (input) ou quitte (output) le système. Nous présentons ici

des LTS montrant les transitions du système entre ses différents états (S, O, P, B). La Figure 7.2 et la Figure 7.3 montrent respectivement les transitions du système lorsque son comportement est contrôlé par les stratégies d'élasticité horizontale et verticale. Notons que les stratégies de migration et de load balancing sont complémentaires à ces méthodes de dimensionnement.

Les LTS proposés ont été modélisés et analysés à l'aide de *LTS-Analyser* [Scalas and Bartoletti, 2015]. Cet outil fournit une plateforme pour définir des processus et des LTS de manière générique afin d'analyser leur ensemble d'états et de vérifier les propriétés de *vivacité* (*progress*) et de *sûreté* (*safety*) liés à leur dynamique. Cet outil permet de vérifier qu'il n'existe pas d'erreurs (deadlock) ou de blocage au niveau des transitions spécifiées. En outre, les LTS montrent que tous les états du système sont accessibles et que chaque état permet d'accéder à un autre à travers les actions d'adaptations définies. Notons que chaque état décrit peut-être l'état initial du système puisque celui-ci est déterminé à l'exécution. Néanmoins, la représentation des LTS à partir de l'état *Stable* comme état initial permet de voir qu'il existe toujours un chemin qui renvoie le système à son état élastique *Stable* même si celui-ci transite à travers les autres états possibles. Cela permet de décrire le bon fonctionnement de l'élasticité des systèmes Cloud en plus de montrer la *non-plasticité* [Bersani et al., 2014] du système en termes d'adaptations élastiques.

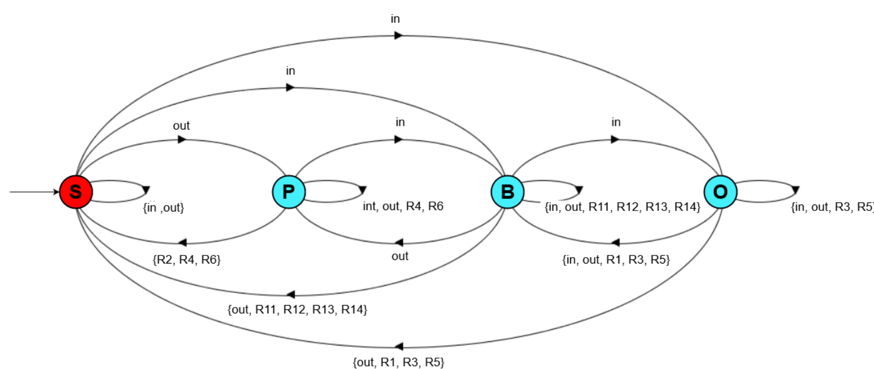


Figure 7.2 – Système de transition pour l'élasticité horizontale

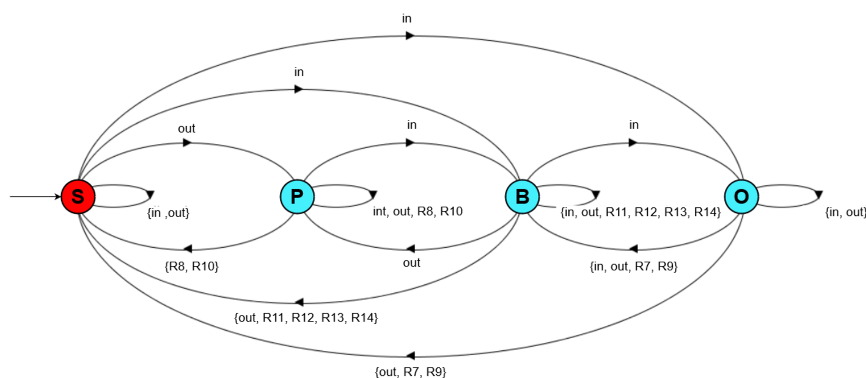


Figure 7.3 – Système de transition pour l'élasticité verticale

Définition des formules propositionnelles LTL. Nous introduisons un ensemble de formules propositionnelle dans la logique temporelle linéaire $LTL(AP_{CS})$ afin de décrire les propriétés dynamiques du système. Il s'agit de décrire les comportements désirables du système en termes de transitions d'états de départ vers des états d'arrivée voulus à l'aide de la sémantique de la logique temporelle linéaire. Les formules propositionnelles LTL sont données dans l'ensemble

$$LTL(AP_{CS}) = \{UpScale, DownScale, Balance, Elasticity\}$$

De la manière suivante :

— $UpScale \equiv G(Overloaded \rightarrow FStable)$

— $DownScale \equiv G(Overprovisioned \rightarrow FStable)$

— $Balance \equiv G(Unbalanced \rightarrow FStable)$

— $Elasticity \equiv G(\sim Stable \rightarrow FStable)$

Où les formules *UpScale/DownScale* précisent que le système Cloud géré à l'état Overloaded/Overprovisioned (surchargé/sur-dimensionné) finira par atteindre son état Stable éventuellement. De la même manière, la formule Balance spécifie qu'un système à l'état Unbalanced (déséquilibré) finira éventuellement par atteindre son état *Stable*. Enfin, la formule *Elasticity* décrit un comportement plus général et spécifie qu'un système qui n'est pas à l'état stable finira par atteindre celui éventuellement. Les symboles G, F, $\rightarrow$ et $\sim$ représentent respectivement les opérateurs temporels *henceforth* (toujours), *eventually* (eventuellement), *implies* (implique) et *not* (négation) de la logique LTL.

Si les LTS précédemment introduits décrivent la dynamique désirée du système, les formules dans la logique LTL permettent d'exprimer formellement cette dynamique. Cela permettra au model-checker de Maude de raisonner sur le comportement du système afin d'analyser son évolution et d'en déterminer le bon fonctionnement, conformément à la sémantique LTL introduite.

7.3.2 Encodage d'états et de propriétés dans Maude

Afin de permettre au model-checker de Maude de raisonner sur l'élasticité des systèmes Cloud de la manière abordée précédemment, il est primordial d'exprimer dans le langage Maude la sémantique introduite par la structure *Kripke* (états et transitions du système) et les formules propositionnelles de la logique LTL (comportements désirés). Pour accomplir cette tâche, Maude permet de définir ces deux spécifications dans un module système, *Elastic_Cloud_Properties*, dédié à la spécification des propriétés du système. Ce module permet de définir les différents états du système ainsi que les formules LTL et fournit une mécanique pour la vérification de leur satisfaction. Ensuite, ce nouveau module sera associé à la théorie de réécriture, définie dans le module système *Elastic_Cloud_Behavior*, décrivant les comportements du modèle spécifié. La Figure 7.4 donne une vue d'ensemble de modules composant notre solution complète pour la spécification, l'exécution et la vérification formelle de l'élasticité des systèmes Cloud dans Maude.

La structure de Kripke est encodée dans le module de spécification des propriétés *Elastic_Cloud_Properties* en deux temps [Clavel et al., 2016]. Nous précisons la sorte (parmi celles déclarées précédemment) qui représentera les états considérés par le model-checker. Ensuite, nous définissons dans Maude les prédicats pertinents représentant les différents états élastiques symboliques (AP_{CS}) via la relation de satisfaction $|=$. Quant

aux formules propositionnelles LTL, elles sont directement encodées en tant que propriétés dans le langage Maude, à travers les équations correspondantes. Les opérateurs G, F, $\rightarrow$ et $\sim$ précédemment utilisés sont respectivement encodé par `[]`, `<>`, `->` et `~`. Le Tableau 7.3 regroupe les principales déclarations du module *Elastic_Cloud_Properties*.

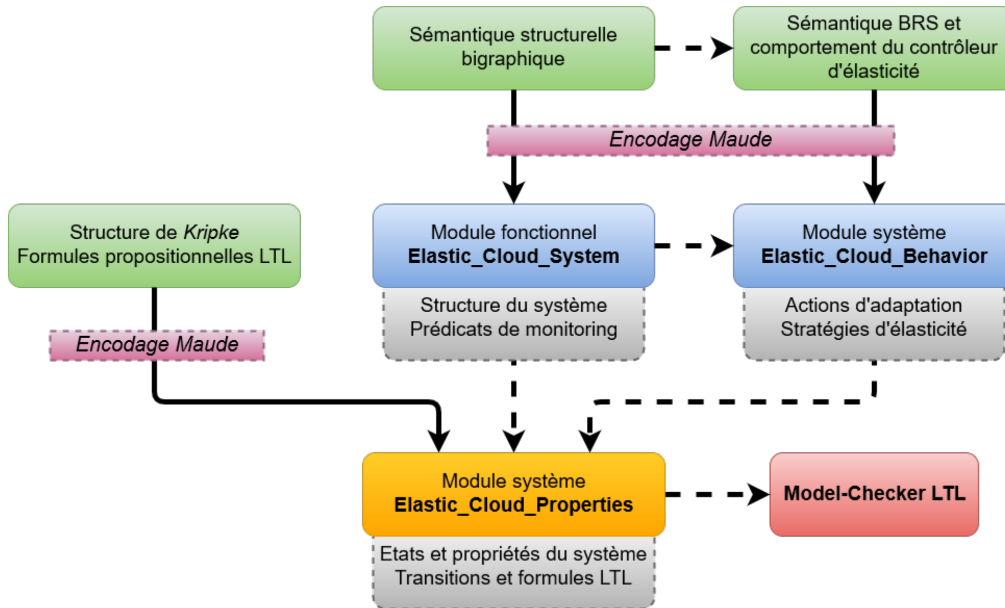


FIGURE 7.4 – Vue d'ensemble de la solution de spécification, d'exécution et de vérification dans Maude

7.3.3 Vérification de propriétés

Une fois le module *Elastic_Cloud_Properties* défini, le model-checker LTL de Maude peut enfin être exécuté afin de vérifier les comportements élastiques et les propriétés spécifiées. Le model-checker est lancé via la commande *modelCheck* et prend en entrée (1) une configuration initiale du système et (2) une propriété sous forme d'une formule LTL à vérifier. En sortie, il retourne la valeur booléenne *True* (vrai) si la propriété est satisfaite, ou un contre-exemple quand elle est violée au cours de l'exécution du système [Clavel et al., 2016]. Le contre-exemple consiste en une trace d'exécution du système, à partir de son état initial, menant à un état où la propriété en entrée a été violée.

Exemple. Afin d'illustrer le fonctionnement du model-checker, considérons l'exemple simple d'un système Cloud *cs* dont nous voudrions vérifier le comportement élastique horizontal. Dans sa configuration initiale, le système Cloud est composé d'un serveur physique hébergeant une machine virtuelle où une instance de service est en exécution. En termes de ressources, la VM dispose d'une unité de CPU et d'une unité de RAM et le serveur dispose de deux unités de CPU et de RAM disponibles. Au niveau de seuils d'hébergement max pour les entités déployées, le système peut accueillir jusqu'à 2 machines virtuelles, une VM jusqu'à 4 instance de service et un service peut gérer jusqu'à 10 requêtes. Cette configuration *cs* est explicitée syntaxiquement selon notre définition des systèmes Cloud du module *Elastic_Cloud_System* de la manière suivante :

```
CS < 2,4,10 / VM{4, S[10,10 : over] : [1 & 1] : stable} : [2 & 2] : over >
```

TABLE 7.3 – Principales déclarations du module *Elastic_Cloud_Properties*

Module de spécification des propriétés *Elastic_Cloud_Properties*

Sortes et opérations

```
subsorts HSCALE VSCALE < State .
subsort ST < HSCALE . subsort ST < VSCALE .
op sysStruct(_) : ST -> CS . .
var cs : CS . vars i j : Nat .
eq sysStruct(HSCALE(V i, S j) :: cs) = cs .
eq sysStruct(VSCALE :: cs) = cs .
```

Prédicats pour les états du système (AP_{CS})

```
ops Stable Overloaded Overprovisioned Unbalanced : -> Prop [ctor] .
var cs : ST . var csh : HSCALE . var csv : VSCALE . var P : Prop .
ceq cs |= Stable = true if isStable(sysStruct(cs)) .
ceq cs |= Overloaded = true if isOverloaded(sysStruct(cs)) .
ceq csh |= Overprovisioned = true if isUnused(sysStruct(csh)) .
ceq csv |= Overprovisioned = true if isSurplus(systStruct(csv)) .
ceq cs |= Unbalanced = true if (LBpred(cs) or MigSpred(cs)) .
ceq cs |= P = false [owise] .
```

Formules propositionnelles ($LTL(AP_{CS})$)

```
ops eq UpScale = [] ( Overloaded -> <> Stable ).
eq DownScale = [] ( Overprovisioned -> <> Stable ).
eq Balance = [] ( Unbalanced -> <> Stable ).
eq Elasticity = [] ( ~ Stable -> <> Stable ).
```

L'instance de service déployée étant initialement surchargée, le système est lui aussi à l'état *Overloaded* puisqu'il est nécessaire qu'il s'adapte pour atteindre son état stable. Le model-checker va vérifier si le système s'adapte correctement à cet état de surcharge, en appliquant les actions d'élasticité horizontale adéquates selon la stratégie d'élasticité horizontale H_Out1 aux niveau infrastructure et application (`HSCALE(V 1, S 1) :: cs`). Le model-checker est exécuté avec la configuration initiale *cs* et la propriété *UpScale* à vérifier. Nous rappelons que cette propriété tente de vérifier qu'il est toujours possible pour un système Cloud d'atteindre l'état d'élasticité *Stable* désiré à partir de l'état *Overloaded* (surchargé). La Figure 7.5 montre le résultat du model-checker avec les entrées mentionnées où le résultat True (vrai) est retourné, indiquant que la propriété a bien été assurée.

```
Maude> reduce in Elastic_Cloud_Properties :
modelCheck(HSCALE(V 1,S 1):: CS< 2,4,10 / VM{4,S[10,10 : over] : [1 & 1]
: stable} : [2 & 2] : over >, UpScale) .
rewrites: 46631 in 16ms cpu (17ms real) (2914437 rewrites/second)
result Bool: true
```

FIGURE 7.5 – Résultat de la vérification de l'élasticité horizontale sous LTL Maude

Tentons maintenant de vérifier la négation de la propriété d'élasticité précédente

(∼*UpScale*) sur le même état initial. La Figure 7.6 montre le résultat de cette vérification où un contre-exemple est affiché. Ce contre-exemple donne la trace d'exécution du système à partir de son état initial. La trace montre que le système s'adapte bien de manière horizontale en déployant une nouvelle instance de service, puis en appliquant de manière successive les actions de load balancing afin d'équilibrer la charge du système au niveau application. Nous représentons l'état initial et l'état final en gras pour facilement les distinguer. Notons que l'état de deadlock à la fin indique qu'aucune action d'adaptation n'est déclenchable à cet état, puisque celui-ci est Stable en termes de son élasticité. L'état final atteint est le suivant :

```
CS< 2,4,10 / VM{4,S[10,5 : stable] + S[10,5 : stable] :
 [1 & 1] : stable} : [2 & 2] : stable >
```

```
Maude> reduce in Elastic_Cloud_Properties : modelCheck(HSCALE(V 1,S 1):: CS< 2,4,10 / VM{4,S[
    10,10 : over] : [1 & 1] : stable} : [2 & 2] :  over >, ~ UpScale) .
rewrites: 8831 in 4ms cpu (3ms real) (2207750 rewrites/second)
result ModelCheckResult: counterexample({HSCALE(V 1,S 1):: CS< 2,4,10 / VM{4,S[
    10,10 : over] : [1 & 1] : stable} : [2 & 2] :  over >,'scale-S1} {HSCALE(V 1,S 1):: CS< 2,4,10 /
    VM{4,S[10,0 : new] + S[10,10 : over] :[1 & 1]: stable} : [2 & 2] : over >,'mark-stable-CS} {
    HSCALE(V 1,S 1):: CS< 2,4,10 / VM{4,S[10,0 : new] + S[10,10 : over] :
    stable} : [2 & 2] : stable >,'load-balancing-S-level} {HSCALE(V 1,S 1):: CS< 2,4,10 /
    VM{4,S[10,1 : new] + S[10,9 : over] : [1 & 1] : stable} : [2 & 2] : stable >,
    'load-balancing-S-level} {HSCALE(V 1,S 1):: CS< 2,4,10 / VM{4,S[10,2 : new]
    + S[10,8 : over] : [1 & 1] : stable} : [2 & 2] : stable >,'load-balancing-S-level} {HSCALE(V 1,
    S 1):: CS< 2,4,10 / VM{4,S[10,3 : new] + S[10,7 : over] : stable} : stable
    >,'load-balancing-S-level} {HSCALE(V 1,S 1):: CS< 2,4,10 / VM{4,S[10,4 :
    new] + S[10,6 : over] : [1 & 1] :  stable} : [2 & 2] : stable >,'load-balancing-S-level} {
    HSCALE(V 1,S 1):: CS< 2,4,10 / VM{4,S[10,5 : new] + S[10,5 : over] : [1 & 1] :
    stable} : [2 & 2] : stable >,'mark-stable-S} {HSCALE(V 1,S 1):: CS< 2,4,10 / VM{4,S[
    10,5 : stable] + S[10,5 : over] : [1 & 1] :  stable} : [2 & 2] : stable >,'mark-stable-S}, {
    HSCALE(V 1,S 1):: CS< 2,4,10 / VM{4,S[10,5 : stable] + S[10,5 : stable] : [1 & 1] :
    stable} : [2 & 2] : stable >,deadlock})
```

FIGURE 7.6 – Contre-exemple résultant de la vérification de l'élasticité horizontale par la négation

De la même manière, nous pouvons soumettre le système *cs* à la vérification formelle de son comportement élastique selon l'élasticité verticale. Le résultat *true* est renvoyé pour la vérification de la propriété UpScale (voir Figure 7.7).

```
Maude> reduce in Elastic_Cloud_Properties :
modelCheck(VSCALE :: CS< 2,4,10 / VM{4,S[10,10 : over] : [1 & 1]
: stable} : [2 & 2] : over >, UpScale) .
rewrites: 45642 in 17ms cpu (18ms real) (2920142 rewrites/second)
result Bool: true
```

FIGURE 7.7 – Résultat de la vérification de l'élasticité verticale sous LTL Maude

Un contre-exemple montrant l'allocation de ressources additionnelles à la VM est renvoyé dans le cas de la vérification de la négation de cette propriété ∼*UpScale* (voir Figure 7.8). Notons qu'allouer plus de ressources à la VM lui permet d'accueillir une instance de service supplémentaire (son seuil *max* sera augmenté de 1). Également, l'instance de service initialement déployée sera en mesure de gérer plus de requêtes, en augmentant de moitié son seuil d'hébergement *max*. Concernant le serveur physique, il pourra accueillir moins de VMs car il disposera de moins d'unités de CPU disponibles (son seuil max sera diminué de 1). La configuration finale atteinte est la suivante :

```
CS< 1,4,10 / VM{5,S[15,10 : stable] : [2 & 1] :stable} : [1 & 2] : stable >
```

```
Maude> reduce in Elastic_Cloud_Properties : modelCheck(VSCALE:: CS< 2,4,10 / VM{4,S[
    10,10 : over] : [1 & 1] : stable} : [2 & 2] :  over >, ~ UpScale) .
rewrites: 9522 in 5ms cpu (4ms real) (2314732 rewrites/second)
result ModelCheckResult: counterexample({VSCALE:: CS< 2,4,10 / VM{4,S[
     10,10 : over] : [1 & 1] : stable} : [2 & 2] :  over >,'addCPU-V}{VSCALE:: CS< 2,4,10 /
     VM{4,S[10,10 : over] : [2 & 1] : stable} : [1 & 2] :  over >,'maxDown-SE}{VSCALE::
     CS< 1,4,10 / VM{4,S[10,10 : over] : [2 & 1] : stable} : [1 & 2] :  over >,'maxUp-V}{
     VSCALE:: CS< 1,4,10 / VM{5,S[10,10 : over] : [2 & 1] : stable} : [1 & 2] :  over >,
     'maxUp-S}{VSCALE:: CS< 1,4,10 / VM{5,S[15,10 : over] :[2 & 1]: stable} : [1 & 2] :
     over >,'mark-stable-CS}{VSCALE:: CS< 1,4,10 / VM{5,S[15,10 : over] :[2 & 1]: stable}
     : [1 & 2] : stable >,'mark-stable-S},{
     VSCALE:: CS< 1,4,10 / VM{5,S[15,10 : stable] : [2 & 1] :stable} :
     [1 & 2] : stable >,deadlock})
```

FIGURE 7.8 – Contre-exemple résultant de la vérification de l'élasticité horizontale par la négation

7.3.4 Bilan

L'étape de vérification formelle des comportements élastiques définis permet dé vérifier leur bon fonctionnement. Cela consiste à assurer que certaines propriétés liées à l'élasticité du système sont vérifiées lors de son exécution. Pour y parvenir, nous avons d'abord défini une structure de Kripke afin d'identifier les différents états et transitions caractérisant l'élasticité du système. Ensuite, nous avons présenté des formules propositionnelles de la logique linéaire temporelle LTL afin de décrire de manière formelle les comportements souhaités du système. Après avoir exprimé les états et les propriétés du système dans Maude, à travers le module *Elastic_Cloud_Properties*, nous avons utilisé le model-checker LTL de Maude afin de conduire la vérification formelle des comportements désirés du système. À l'issu de cette étape, nous avons donc atteint notre troisième objectif de recherche (OR3) visant à fournir un moyen de vérifier sur le plan qualitatif la dynamique comportementale liée à l'élasticité des systèmes Cloud (voir Section 5.1).

7.4 Conclusion

Dans ce chapitre, nous avons présenté un encodage des spécifications bigraphiques et des stratégies d'élasticité dans le langage de spécification formelle Maude. Cet encodage permet de préserver la sémantique structurelle des systèmes Cloud tout en l'enrichissant d'aspects quantitatifs et la possibilité d'exprimer les états du système à travers des prédicats de la logique du premier ordre. Maude permet également d'encoder les stratégies d'élasticité afin de permettre leur exécution de manière générique et autonomique. Nous avons procédé à la vérification formelle de l'élasticité des systèmes Cloud modélisés, grâce aux outils fournis par le système Maude. Cette vérification se base sur une technique de model-checking à base d'états, supportée par la logique temporelle linéaire LTL et les structures de *Kripke*.

Dans le chapitre suivant, nous introduisons une approche d'évaluation et de validation quantitative de l'élasticité d'un système Cloud. Nous présentons une étude expérimentale d'un système Cloud opérant selon les comportements élastiques que nous avons défini.

Chapitre 8

Évaluation quantitative des stratégies d'élasticité multi-couches

Sommaire

8.1 Introduction 120
8.2 Étude de cas : modélisation de la plateforme Steam ® 121
8.3 Évaluation outillée de l'élasticité à base de files d'attente 124
8.3.1 Modèle de files d'attente 124
8.3.2 Principes de la simulation à base de files d'attente 126
8.3.3 Un outil pour la simulation de l'élasticité dans le Cloud 129
8.4 Simulation et évaluation des comportements élastiques de la plateforme Steam 131
8.4.1 Paramétrage des simulations 131
8.4.2 Stratégies de dimensionnement Horizontal 133
8.4.3 Migration et Load Balancing 140
8.4.4 Stratégies de dimensionnement Vertical 140
8.4.5 Comparaison de l'élasticité Horizontale et Verticale 143
8.5 Conclusion et discussion 144

8.1 Introduction

Dans les précédents chapitres de contributions, nous avons présenté notre approche de modélisation des systèmes Cloud pour la formalisation et la vérification de leurs comportements élastiques. Nous avons introduit un ensemble de stratégies d'élasticité (i.e., horizontale, verticale, migration et load-balancing) opérant en multi-couches (i.e., infrastructure et application). Ces stratégies décrivent la logique du contrôleur d'élasticité qui gère l'élasticité d'un système Cloud contrôlé de manière autonomique. Le contrôleur surveille l'état et l'activité du système, et déclenche les actions d'adaptation nécessaires (i.e., ajout, retrait, migration de ressources) selon une ou plusieurs stratégies, dans le but de maintenir une bonne qualité de service du système Cloud contrôlé, et d'éviter les états de sur-dimensionnement et sous-dimensionnement des ressources .

Les stratégies d'élasticité permettent au contrôleur d'élasticité de prendre des décisions concernant l'exécution des mécanismes d'élasticité. Précisément, elles permettent de décider quand, où et comment déclencher les actions liés aux différentes méthodes d'élasticité définies. Ces stratégies sont chargées de garantir l'allocation des ressources nécessaires et suffisantes pour assurer le bon fonctionnement du du système Cloud contrôlé en dépit des

fluctuations dans sa charge de travail en entée. Afin de garantir l'efficacité des stratégies définies, il est primordial de les évaluer et de les valider du point de vue quantitatif, avant de les utiliser dans des environnement Cloud réels.

Dans ce dernier Chapitre dédié aux contributions, nous illustrons notre approche de modélisation et de vérification formelle à travers une étude de cas d'un système Cloud élastique. Nous proposons un support outillé pour évaluer et analyser les comportements élastiques définis à travers une étude expérimentale par simulation. L'objectif principal des expérimentations que nous allons présenter ici est de mettre en avant les apports des différentes stratégies introduites dans ce manuscrit et d'en analyser les répercussions sur le système étudié sur le plan quantitatif. Dans la Section 8.2, nous introduisons le système étudié et montrons comment un tel système peut être modélisé à l'aide des systèmes réactifs bigraphique, puis encodé dans le langage Maude. Nous discutons également les possibilités de vérification formelles d'un tel système. Dans la Section 8.3, nous expliquons les principes de la simulation à base de files d'attente et introduisons un outil conçu à cet effet. Dans la Section 8.4, nous étudions la dimension quantitative de l'élasticité multi-couches du système analysé, selon les stratégies d'élasticité définies. Nous proposons une approche à base de files d'attente pour simuler, évaluer et valider les comportements élastiques autonomiques d'un système Cloud. Dans la Section 8.5 nous concluons enfin ce Chapitre par une discussion sur les différents aspects entrant en jeu afin de contrôler l'élasticité d'un système Cloud.

8.2 Étude de cas : modélisation de la plateforme Steam ®

Steam est une plateforme de distribution de contenu en ligne, de gestion des droits et de communication développée par Valve en 2003 [Steam, 2019]. Principalement orientée autour des jeux vidéo, la plateforme Steam permet aux utilisateurs d'acheter des jeux, du contenu pour les jeux et de les mettre à jour automatiquement. En Janvier 2018, Steam comptait près de 125 Millions d'utilisateurs enregistrés et un catalogue de près de 28000 produits en vente. Pour l'utilisateur, la boutique *Steam* se présente comme n'importe quel site de commerce électronique sur internet. Elle est accessible depuis un navigateur internet classique ou par le biais d'un logiciel client sur ordinateur ou mobile. Depuis 2013, Steam bénéficie d'un support basé Cloud complet pour l'hébergement des différents services offerts. Nous nous intéressons dans notre étude de cas à la boutique de vente en ligne de Steam.

Steam base l'essentiel de son modèle économique sur la haute disponibilité de ses services et une accessibilité permanente et performante pour tous les pays du monde. D'après le site web [SteamSpy, 2019], près d'un million de produits ont été vendus sur Steam en 2018, avec une moyenne de 83.000 ventes par mois. Si la boutique fait l'objet d'une sollicitation permanente, des pics de ventes sont enregistrés pendant les évènements saisonniers tels que les soldes et remises de fin d'année. Par exemple, près de 200.000 ventes ont été enregistrées pendant la dernière semaine de décembre 2017.

De ces chiffres, nous pouvons observer que le service de la boutique en ligne *Steam* connait une activité importante (hors consultations, recherches, affichage de contenus, transactions annulées, etc.) et de nature hautement variable. Cela en fait un bon exemple pour évaluer notre solution pour la gestion des comportements élastiques d'un système Cloud. Dans la sous-section suivante, nous appliquons notre approche de modélisation et de vérification formelle du service de vente en ligne *Steam*. Ensuite, nous proposons une

étude expérimentale de l'élasticité de ce service afin de valider quantitativement notre modèle proposé.

Application de l'approche de modélisation formelle

Le service de la boutique en ligne Steam est basé sur un système de type Cloud. De ce fait, il peut facilement être pris en charge par notre approche générique de modélisation formelle. Dans un premier temps, nous modélisons de manière bigraphique, puis conformément à la syntaxe de Maude une configuration de ce système. Ensuite, nous vérifions formellement les différents comportements élastiques du système à partir de cette configuration initiale.

Pour appliquer notre approche de modélisation, considérons une configuration du système où le service de vente en ligne est déployé au sein d'une machine virtuelle, elle-même déployée au sein d'un serveur physique. Nous supposerons que le serveur dispose de 4 unités de CPU et de RAM et que la VM dispose d'une unité de CPU et de RAM.

Modélisation bigraphique. la Figure 8.1 donne la configuration du système obtenue, en appliquant la modélisation selon le sémantique bigraphique et sa logique de typage introduites dans le Chapitre 6. Le nœud de contrôle SE modélise le serveur physique, le nœud VM la machine virtuelle déployée et le nœud S modélise une instance du service de vente en ligne Steam. Concernant les ressources informatiques déployées, les nœuds de contrôle CU et M modélisent les unités de CPU et de RAM respectivement. Les pools de ressources disponibles au niveau du serveur physique et de la VM sont modélisés par les nœuds de contrôle RD et RV respectivement. Le nom externe w (pour *workload*) connecté au serveur physique, sert à représenter l'interfaçage du système avec son environnement extérieur (clients finaux). Nous rappelons que les sites (numérotés de 0 à 5 ici) servent à abstraire certaines parties et éléments du système. Par exemple, le site 5, placé au sein de l'instance de service S, sert à abstraire les requêtes gérées par cette instance.

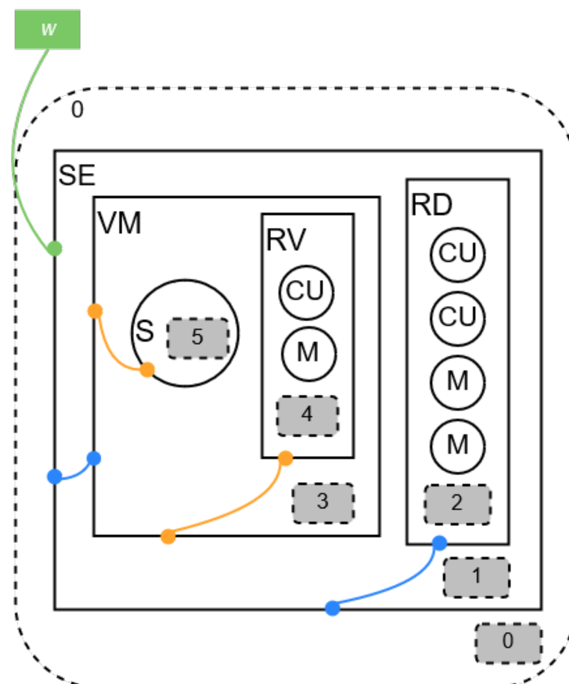


FIGURE 8.1 – Représentation bigraphique d'une configuration du service de vente en ligne Steam

Encodage dans Maude. En appliquant l'encodage basé sur le langage Maude à la configuration du système proposée, nous obtenons la même structure enrichie de plusieurs aspects. En effet, Il devient possible de représenter certaines notions telles que les seuils max initiaux en termes de capacités d'hébergement pour le serveur physique, les VMs et les instances de services déployés. Ces seuils sont représentés par les variables x, y et z respectivement. De plus, nous représentons l'état de ces instances du point de vue de l'élasticité, comme montré dans la Section 6.3. La forme générale de la configuration du service Steam basé Cloud étudié est donnée comme suit :

```
CS< x,y,z/ VM{y',S[z',q : stateS]:[1 & 1]: stateVM}:[2 & 2]: stateSE >
```

Où les termes *stateS*, *stateVM* et *stateSE* représentent les états (*overloaded*, *unused*, *stable*, etc.) de l'instance de service, de la VM et du serveur physique respectivement. Les variables y' et z' représentent les seuils *max* de la VM et du service S s'ils sont différents des valeurs initiales. La variable q représente le nombre de requêtes gérées par S. Enfin, les structures `[a & b]` représentent les ressources informatiques déployées où a et b donnent le nombre d'unités de CPU et de RAM respectivement.

Vérification formelle de l'élasticité. Dans la Section 7.3, nous avons décrit le processus de vérification formelle de l'élasticité dans le Cloud à travers les différentes stratégies définies. Nous utilisons une technique de model-checking à base d'états supportée par la logique temporelle linéaire LTL. Le model-checker de Maude est exécuté avec, en entrée, une configuration initiale du système et une propriété (formule propositionnelle LTL) à vérifier. Nous avions défini quatre propriétés dans LTL (*UpScale*, *DownScale*, *Balance* et *Elasticity*) pour décrire le comportement élastique désiré d'un système donné. Comme configuration initiale, reprenons celle du service Steam introduite pour la modélisation bigraphique. Une fois adaptée selon l'encodage Maude, il nous faut définir les seuils max pour les différentes entités ainsi qu'un nombre de requêtes traitées par l'instance de service.

Notons qu'il est possible de vérifier l'élasticité du système de 80 manières différentes. Précisément, en spécifiant l'état du système (surchargé/sur-dimensionné/stable/déséquilibré), selon quelle stratégie d'élasticité celui-ci se comporte (horizontale/verticale) et en donnant une des quatre propriétés LTL à vérifier :

4 états $\times$5 possibilités de dimensionnement $\times$4 propriétés = 80 possibilités de vérification.

Les cinq possibilités de dimensionnement viennent du fait que le dimensionnement horizontal (ScaleOut) puisse s'appliquer de deux manières différentes (disponibilité élevée/limitée) et à deux niveaux différents (infrastructure et application) du système Cloud. Ce qui donne quatre combinaisons « *cross-layer* » ou multi-couches possibles, en plus du dimensionnement vertical. Parmi les combinaisons possibles, certaines sont plus pertinentes pour la vérification. Par exemple, il est plus intéressant de soumettre, au model-checker, une configuration du système qui serait surchargée afin de vérifier la propriété *UpScale*. Réciproquement, la vérification de la propriété *DownScale* prend tout son sens si la configuration vérifiée est à l'état *Overprovisioned* (sur-dimensionné). La propriété *Balance* est plus adéquate à la vérification pour une configuration à l'état *Unbalanced* (déséquilibré). Enfin, la propriété *Elasticity* peut être utilisée dans tous les cas du fait qu'elle décrive un comportement général (voir Section 7.3).

La vérification formelle de l'élasticité s'assure que le système Cloud adopte un comportement désirable durant son fonctionnement. Néanmoins, la dimension quantitative liée à l'élasticité telles que la performance, le temps de réponse, les fluctuations de la demande et son impact ou encore les coûts liés aux ressources déployées reste non abordée jusqu'à présent. Dans la Section suivante, nous présentons une étude expérimentale menée en vue d'évaluer et de valider quantitativement les comportements élastiques définis.

8.3 Évaluation outillée de l'élasticité à base de files d'attente

L'élasticité vise à gérer d'une manière précise l'ajout et le retrait de ressources Cloud afin d'ajuster leur disponibilité, en fonction de la demande (voir Section 2.3). Cela revient à éviter les cas de *sur-dimensionnement* et de *sous-dimensionnement* qui impactent aussi bien les fournisseurs que les usages du service, en termes de coûts et de performance. Dans la Section 6.3, nous avons défini plusieurs stratégies d'élasticité visant à guider le comportement autonomique d'un système Cloud élastique. En appliquant les différentes stratégies introduites, un système Cloud peut adopter des comportements différents, qui se traduisent par des adaptations et des dimensionnements différents.

Dans cette section nous appliquons une approche basée sur la théorie des files d'attente [Kleinrock, 2005] afin d'étudier et d'évaluer les différentes stratégies définies (voir Section 2.3). Nous analyserons l'élasticité du service de la boutique en ligne de la plateforme Steam. Dans un premier temps, nous proposons un modèle de files d'attente afin de modéliser les principaux paramètres et métriques considérés dans notre étude. Ensuite, nous montrons comment le modèle défini peut être utilisé dans le but d'analyser les comportements élastiques d'un système donné. Enfin, nous simulerons le système étudié à travers plusieurs scénarios afin d'analyser son comportement élastique.

8.3.1 Modèle de files d'attente

Un modèle de file d'attente permet de représenter les systèmes de files d'attente [Baynat, 2000]. Un système de file d'attente consiste en des clients qui arrivent dans le système afin de recevoir un service quelconque. Quand les serveurs (offrant le service) sont occupés, les clients attendent en file puis quittent le système une fois servis. En appliquant cette vision sur un système informatique basé Cloud, les clients qui arrivent dans le système prennent la forme de requêtes reçues de la part des utilisateurs finaux, et les serveurs sont représentés par les instances du service déployées au sein des différentes machines virtuelles disponibles.

Afin de décrire et d'analyser le fonctionnement de la boutique en ligne Steam sous le prisme d'un système de file d'attentes, nous définissons un modèle de file d'attente selon la notation de Kendall [Kendall, 1953].

Paramètres de l'évaluation. La notation de Kendall introduit les paramètres A/S/K/-Q/P/D où :

— A est le processus d'arrivée. Il donne le nombre de requêtes en entrée dans le système par une distribution aléatoire suivant la loi de Poisson, autour d'un taux moyen λ.

— S est le processus de service. Il donne le nombre de requêtes traitées par une distribution aléatoire exponentielle, autour d'un taux moyen μ.

— K est le nombre total d'instances du service de la boutique en ligne Steam, déployées au sein des différentes VMs disponibles.

— Q est la taille de la file d'attente du système où les requêtes reçues sont regroupées avant d'être transmises aux différentes instances de service disponibles.

— P est nombre de clients potentiels du service de la boutique en ligne Steam.

— D est la discipline de service (e.g. FIFO, SJF, etc.) décrivant l'ordonnancement du traitement des requêtes.

En d'autres termes, les utilisateurs finaux de la boutique en ligne représentent la population potentielle P du système. Les utilisateurs envoient des requêtes (consultation, recherche, achat, etc.) qui arrivent dans le système selon une affluence décrite par le processus d'arrivée A. Les requêtes sont enfilées dans la file d'attente du système de taille Q, pouvant être finie ou infinie. Ensuite, les requêtes sont réparties entre les K différentes instances du service déployées. Enfin, les requêtes sont traitées selon une discipline d'ordonnancement donnée avant de quitter le système. Le processus de service S donne, pour chaque instance de service, le nombre moyen de requêtes traitées par unité de temps.

En appliquant l'approche des files d'attente sur notre modélisation des systèmes Cloud élastiques, il est intéressant d'observer que le nombre K d'instances de service dépend du dimensionnement du système en machines virtuelles. En effet, une VM héberge un nombre d'instances de service borné par sa capacité d'hébergement *max*. Étant donné que le nombre de VM est lui aussi borné par la capacité *max* du serveur physique, le nombre maximal de services déployés est en réalité borné. De la même manière, le nombre de requêtes prises en charge par une instance de service peut être considéré comme une file d'attente bornée par le seuil d'hébergement *max* pour chaque instance. Ainsi, la capacité du système à satisfaire les requêtes en entrée est déterminée par le nombre K d'instances de service disponibles.

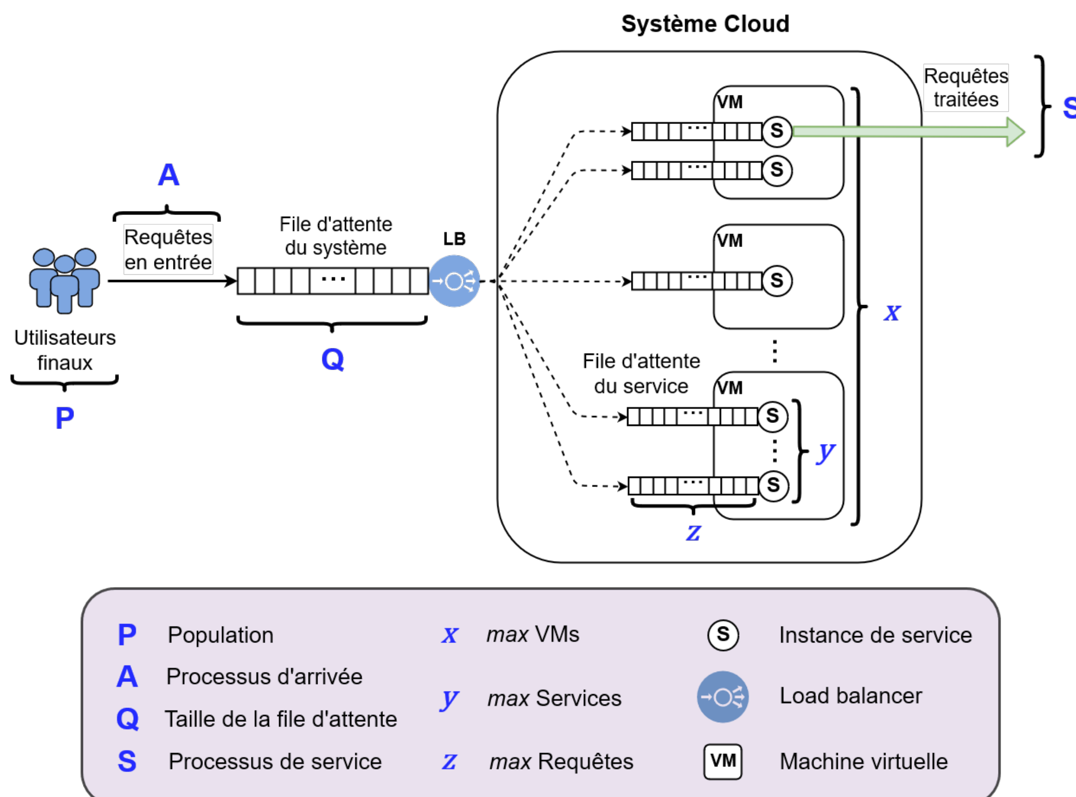


FIGURE 8.2 – Principe d'application de l'approche file d'attente sur les systèmes Cloud

la Figure 8.2 illustre le principe de la vision à base de files d'attente sur notre modèle de systèmes Cloud. Notons qu'une telle approche permet de modéliser le côté *front-end* d'un système Cloud. En outre, elle permet de représenter les interactions *front-end/back-end* (réception et traitement des requêtes) à l'aide d'une sémantique mathématique robuste (processus d'arrivée et de service, nombre de serveurs, taille de la file d'attente, etc.). Le composant load balancer, point de connexion entre le système cloud et son environnement extérieur, décrit comment les requêtes sont transmises de la file d'attente du système aux différentes instances de service disponibles. Les valeurs x, y et z correspondent aux seuils *max* des serveurs, des VMs et des services introduits précédemment (cf. Section 6.3).

8.3.2 Principes de la simulation à base de files d'attente

Avec les différents paramètres introduits, il est possible de simuler l'exécution d'un système Cloud de plusieurs manières. Notre outil de simulation permet de décrire une infinité de situations et de scénarios d'exécution de n'importe quel système Cloud. Néanmoins, Dans les systèmes à base de files d'attente, on dit souvent que les congestions se produisent quand le nombre de requêtes accroit soudainement, résultant ainsi en des temps d'attente importants [Gueroui, 2015]. En réalité, dans le cas d'une affluence constante et relativement homogène des requêtes, le taux de service μ et la taille Q de la file d'attente peuvent avoir un grand impact sur les performances du système.

Impact du taux de service μ. Le taux de service donne le nombre de requêtes traitées en une unité de temps. De ce fait, μ est un indicateur de performance et peut aussi bien représenter la nature du service concerné. Par exemple, le service d'une API en ligne qui donnerait des informations simples (e.g. informations météorologiques) serait en mesure de traiter un grand nombre de requêtes en un temps minime. Cependant, un service de conversion de fichiers en ligne, par exemple, traiterait les tâches reçues avec des temps d'attente plus importants. Notons que ces notions informelles sont primordiales pour assurer une bonne évaluation des performances d'un système. Définir un temps d'attente ≪ acceptable ≫ dépend fortement de la nature du service. Pour illustrer l'impact du taux de service, la Figure 8.3 montre des traces d'exécution de la boutique en ligne Steam pendant 200 unités de temps. En entrée de la simulation, nous donnons une arrivée constante de requêtes selon le taux moyen $\lambda = 100$ et le seuil maximal de $z = 100$ pour les instances de service en termes de requêtes prises en charge. La capacité du système est donnée par le nombre de requêtes traitées par unité de temps. L'impact de μ sur les performances est donné pour deux valeurs différentes.

La trace (a) montre l'état du système pour $\mu = 50$. La capacité du système étant plus faible que l'affluence des requêtes en entrée, le nombre de requêtes en attente dans la file d'attente du système accroit progressivement jusqu'à la saturation. Cela se traduit par des temps de réponse très importants et des performances médiocres. La trace (b) montre l'évolution du système pour $\mu = 120$. La capacité du système étant plus importante, le nombre de requêtes en attente est plus contrôlé avec 80 requêtes en attente à chaque unité de temps contre 5000 pour la trace (a). En termes de performance, le monitoring enregistre 100% de requêtes traitées avec succès pour la trace (b) contre 49% pour la trace (a).

Impact de la taille de la file d'attente Q. Dans nos simulations, les requêtes en entrée au système sont regroupées dans la file d'attente du système avant d'être transmises aux instances de service pour être traitées. La taille Q de la file d'attente peut aussi bien être finie qu'infinie. Lorsque celle-ci est limitée, les requêtes en entrée sont enfilées jusqu'à sa-

turation de la file d'attente et les requêtes en surplus sont ainsi rejetées (perdues). Dans les cas des traces (a) et (b), où Q est infini, les requêtes sont enfilées indéfiniment jusqu'à leur traitement, ce qui peut résulter en des temps de réponse très importants. Limiter la taille Q peut s'avérer efficace afin d'améliorer les performances du système. Pour $Q = 50$, la trace (c) montre des performances bien supérieurs à celles obtenus dans la trace (a). En effet, le nombre moyen de requêtes en attente de traitement est de 8 requêtes/unité de temps contre 5000 pour la trace (a) pour le même taux de requêtes traitées avec succès (49%). Dans ce cas de figure, 51% des requêtes reçues sont rejetée afin d'éviter la saturation du système. Dans la trace (d), le système atteint des performances maximisées avec 0 requête en attente de traitement. Cependant, 39% des requêtes en entrée sont rejetées. Ainsi, 61% des requêtes reçues sont traitées avec succès.

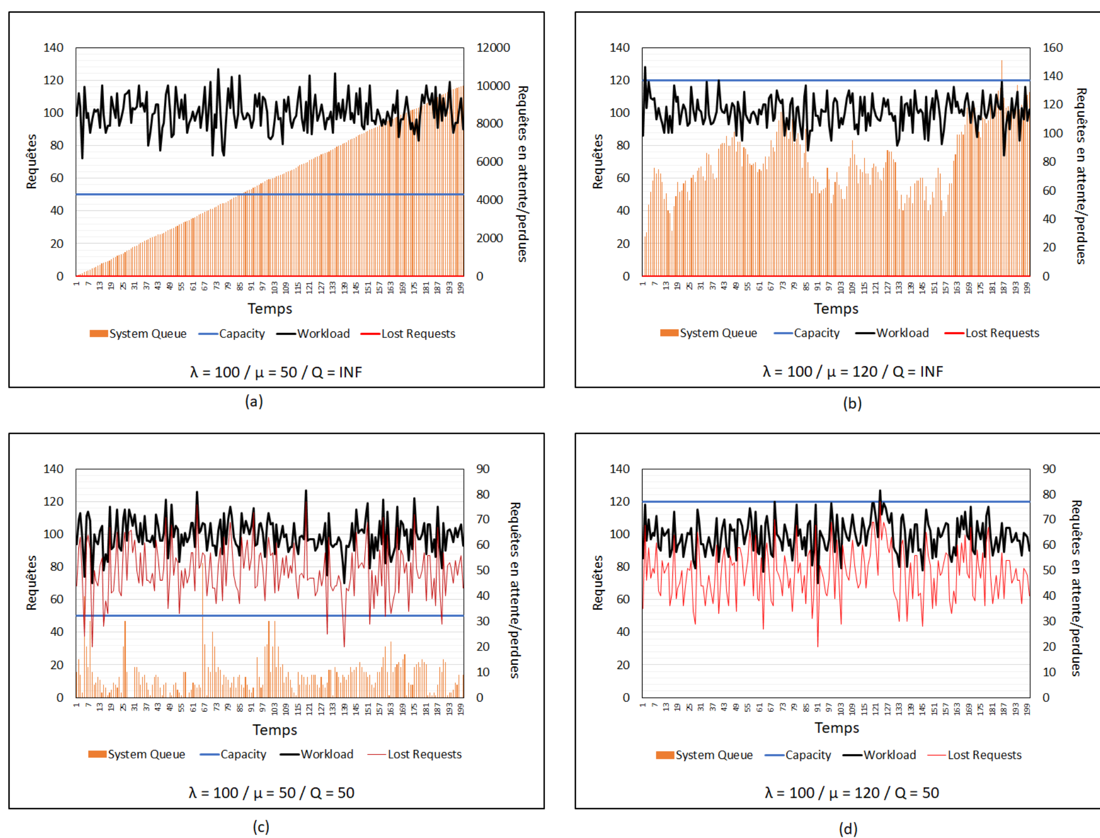
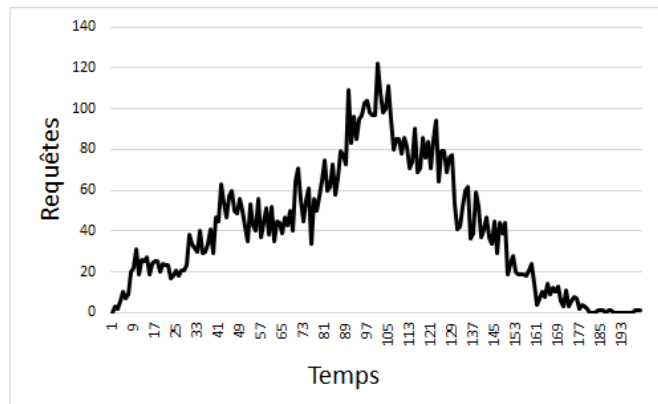


FIGURE 8.3 – Impact du taux de service et de la taille de la file d'attente

Modèles de charge de travail en entrée (*workload*). Jusqu'à présent, nous avons simulé l'exécution d'un système Cloud avec un taux d'arrivée de requêtes constant. Afin d'analyser l'élasticité de nos systèmes Cloud, il est important de disposer de modèles variables au niveau de l'intensité de la charge de travail (requêtes en entrées). En effet, l'élasticité prend tout son sens lorsque le système est confronté à une sollicitation variable qui requiert un redimensionnement (ajout/retrait) des ressources disponibles. Il existe de nombreux modèles décrivant la nature de la charge de travail en entrée du système tels que les modèles périodiques, les modèles continuellement changeants ou encore les modèles imprévisibles [Fehling et al., 2014]. Dans nos simulations, nous utiliserons un modèle imprévisible de workload. Un tel modèle est indépendant de toute fonction le définissant et peut être complètement aléatoire. Néanmoins, nous définissons une allure commune

pour nos différentes simulations afin de comparer les différents comportements élastiques face à des situations similaires. À partir d'un taux d'arrivée λ donné, nous décrivons des valeurs oscillantes au cours de la simulation afin de générer une charge de travail comme montré dans la Figure 8.4. La partie (a) montre l'algorithme permettant de produire la trace de workload de la partie (b). Les résultats sont montrés pour $\lambda = 100$, et 200 unités de temps de simulation. Nous rappelons que le nombre de requêtes est calculé par une distribution aléatoire suivante la loi de Poisson. De ce fait, il est impossible d'obtenir les mêmes valeurs pour les simulations conduites. Néanmoins, nous obtenons des courbes de workload ayant la même allure et décrivant des affluences de requêtes équivalentes.

```
if (tick == 2) lambda = (lambdaMax * 0.05);
else if (tick == 4) lambda = (lambdaMax * 0.1);
else if (tick == 8) lambda = (lambdaMax * 0.2);
else if (tick == 10) lambda = (lambdaMax * 0.25);
else if (tick == 20) lambda = (lambdaMax * 0.2);
else if (tick == 30) lambda = (lambdaMax * 0.35);
else if (tick == 40) lambda = lambdaMax * 0.5;
else if (tick == 50) lambda = lambdaMax * 0.45;
else if (tick == 70) lambda = (lambdaMax * 0.55);
else if (tick == 80) lambda = lambdaMax * 0.7;
else if (tick == 90) lambda = lambdaMax;
else if (tick == 100) lambda = lambdaMax * 0.95;
else if (tick == 110) lambda = lambdaMax * 0.8;
else if (tick == 130) lambda = lambdaMax * 0.5;
else if (tick == 140) lambda = lambdaMax * 0.4;
else if (tick == 150) lambda = (lambdaMax * 0.2);
else if (tick == 160) lambda = (lambdaMax * 0.1);
else if (tick == 170) lambda = (lambdaMax * 0.05);
else if (tick == 180) lambda = (lambdaMax * 0.01);
else if (tick == 190) lambda = (lambdaMax * 0.001);
workload = Poisson.getPoisson(lambda);
```



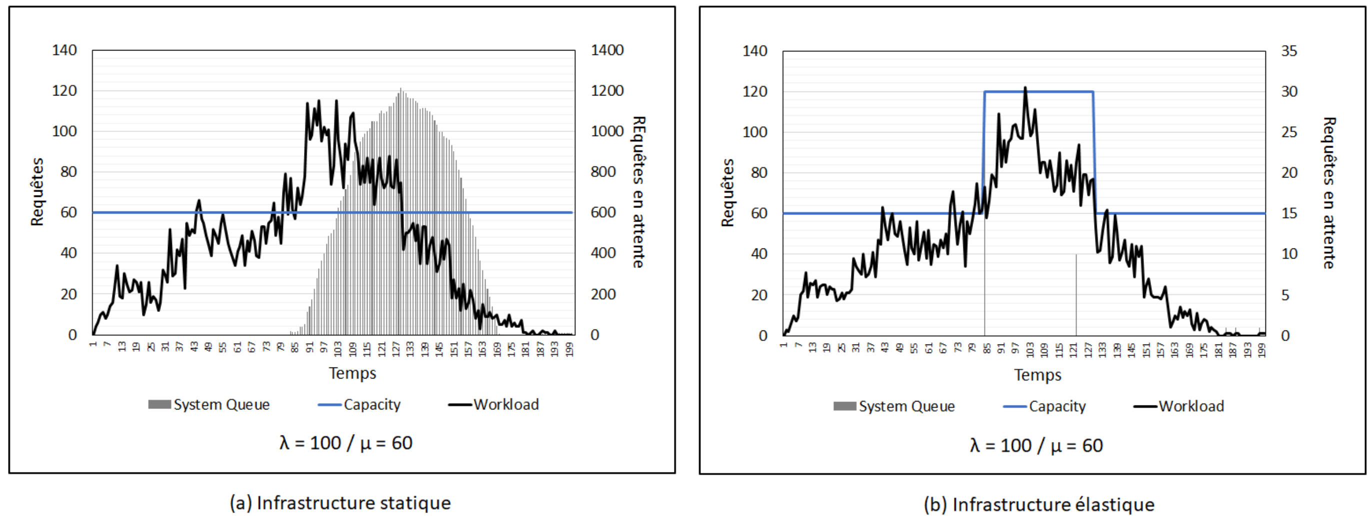
(a) (b)

FIGURE 8.4 – Modèle de workload imprévisible

Infrastructures statiques vs. infrastructures élastiques. Les systèmes Cloud élastiques se distinguent des autres modèles informatiques par leur capacité d'adaptation à leur charge de travail, en ajoutant et en retirant des ressources Cloud, lorsque la demande augmente et diminue. En revanche, les infrastructures dites classiques disposent d'une capacité statique en termes de ressources déployées. la Figure 8.5 montre les résultats de la simulation d'une infrastructure statique (a) et celle d'une infrastructure Cloud élastique (b). Les systèmes sont simulés pour $\lambda = 100$, $\mu = 60$ et pendant 200 unités de temps. Le modèle de workload utilisé correspond à celui expliqué précédemment. Les résultats montrent que l'infrastructure statique maintient une capacité constante tout au long de son exécution. Lors de l'augmentation de la charge de travail enregistrée, la courbe (a) montre que le système est saturé avec une moyenne de 314 requêtes en attente et un pic de 1200 requêtes dans la file d'attente. D'un autre côté, la courbe (b) montre le comportement de l'infrastructure Cloud élastique. Disposant de la même capacité initiale en ressources que l'infrastructure statique, l'infrastructure Cloud s'adapte lorsque sa charge de travail augmente. Cette adaptation consiste à augmenter sa capacité en approvisionnant plus de ressources. Par conséquent, le système assure d'excellentes performances avec quasiment aucune requête en attente. Néanmoins, nous observons des requêtes en attente lorsque la charge de travail augmente soudainement. Lorsque le workload diminue, l'infrastructure Cloud s'adapte afin de baisser sa capacité en ressources. Cela consiste en un redimensionnement en vue de libérer les ressources Cloud superflues.

Notons qu'un système Cloud élastique peut s'adapter à sa demande de plusieurs

manières différentes. En effet, les stratégies d'élasticité introduites dans le Chapitre 6.3 permettent de contrôler l'élasticité d'un système Cloud de manière à satisfaire des politiques de haut niveau différentes. Dans la Section 8.4, nous appliquons notre approche d'évaluation quantitative sur les stratégies introduites, afin d'étudier et de valider leur apport en termes de coûts et de performance.



(a) Infrastructure statique

(b) Infrastructure élastique

FIGURE 8.5 – Infrastructures statiques vs. infrastructures élastiques

8.3.3 Un outil pour la simulation de l'élasticité dans le Cloud

Dans notre évaluation, nous nous inspirons des modèles de files d'attente avec nombre variable de serveurs [Mazalov and Gurtov, 2012], afin d'ajuster le nombre de ressources aux niveaux infrastructure (VMs) et application (instances de service) d'un système Cloud. Précisément, nous mettons à l'œuvre les différentes stratégies d'élasticité définie (voir Section 6.3) afin d'évaluer les performances et les coûts résultants sur le service de la boutique en ligne Steam. À ces fins, nous avons conçu un outil de simulation et de monitoring pour les systèmes Cloud élastiques.

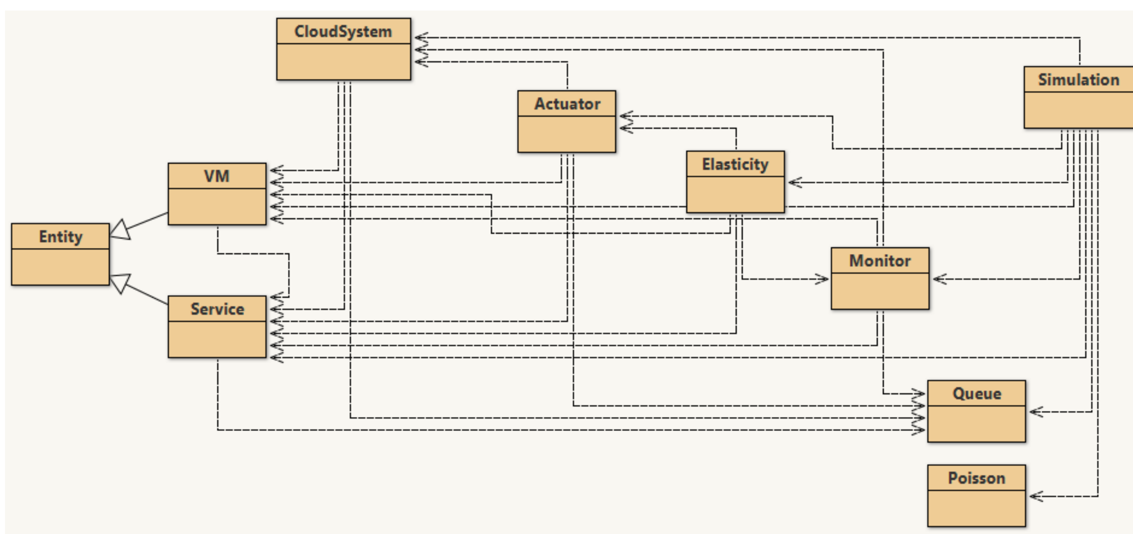


FIGURE 8.6 – Structure fonctionnelle de l'outil de simulation de l'élasticité

Cet outil permet de reproduire le fonctionnement de la boucle de contrôle autonomique MAPE-K implémentant le contrôleur d'élasticité d'u système Cloud (voir Sous Section 2.2.6). Cela permet de surveiller l'évolution de l'état du système Cloud géré en y appliquant les stratégies d'élasticité définies précédemment. L'outil permet de générer un trafic de requêtes en entrée et en sortie du système tout en implémentant le modèle de file d'attente introduit. La structure fonctionnelle de l'outil de simulation est donnée dans la Figure 8.6. L'outil reprend les principales entités architecturales identifiées précédemment pour les systèmes Cloud (voir Chapitre 6) en y intégrant les composantes de la boucle MAPE-K. À savoir, les entités *Monitor*, *Actuator* et *Elasticity* qui représentent respectivement les phases de monitoring et d'action ainsi que le contrôleur d'élasticité en lui-même. D'un autre côté, l'outil intègre une entité *Simulation* pour le paramétrage et l'exécution et l'enregistrement des simulations. Il prend également en compte la notion de files d'attente via l'entité Queue ainsi que l'entité *Poisson*, qui servira pour le calcul des distributions selon la loi de Poisson.

Métriques de l'évaluation. Nous simulons l'exécution du service Steam avec en entrée : (1) une configuration initiale du système Cloud, (2) des valeurs pour les seuils d'hébergement max aux niveaux infrastructure et application, (3) un ou plusieurs taux d'arrivée de requêtes λ et (4) un taux de service μ. Nous rappelons que les taux λ et μ déterminent les processus d'arrivée et de service des requêtes (*input / output*). À chaque unité de temps durant la simulation du système, le monitoring enregistre les informations suivantes :

- Nombre de requêtes en entrée
- Nombre de requêtes en attente dans la file d'attente du système
- Nombre d'instances de service et de VMs déployées
- Nombre de requêtes prises en charge (dans les files d'attente des services)
- Nombre de requêtes traitées avec succès
- Nombre de requêtes perdues

À la fin de la simulation, les informations et métriques suivantes sont calculées, à partir des traces d'exécution du système :

- Nombre moyen d'instances de service et de VMs déployées
- Attente moyenne des requêtes avant traitement
- Taux d'utilisation moyen des instances de service et des VMs
- Taux moyen de requêtes traitées avec succès
- Taux moyen de requêtes perdues

Les métriques obtenues sont utilisées pour indiquer la performance et les coûts liés à l'élasticité du système Cloud simulé. La performance du système est principalement donnée par l'attente moyenne des requêtes avant leur traitement (temps de réponse), ainsi que par le taux moyen de requêtes traitées et perdues. Les coûts et la précision d'adaptation du système sont déterminés à partir du nombre moyen de ressources Cloud (VMs et services) déployées. Enfin, l'efficacité du système est obtenue en analysant la relation entre la performance et les coûts du système ainsi que par le taux d'utilisation moyen des ressources. Afin de valider les résultats obtenus en termes de coûts et d'efficacité, nous comparons ces résultats avec ceux obtenus via la formule d'Erlang-C [Chan, 2014]. À partir des taux d'arrivée λ et de service μ et d'un temps d'attente dit « acceptable », la formule d'Erlang-C calcule le nombre minimal de serveurs (instances de service) nécessaires pour assurer un niveau de service donné. Le niveau de service est donné le nombre de requêtes ayant été servie dans les délais du temps d'attente acceptable.

8.4 Simulation et évaluation des comportements élastiques de la plateforme Steam

Dans le Chapitre 6 , nous avons introduit un nombre de stratégies d'élasticité décrivant le comportement adaptatif d'un système Cloud. Les stratégies introduites sont de nature réactive, prenant la forme *SI conditions ALORS* action. En outre, elles définissent des comportements multi-couches, c'est-à-dire qu'elles peuvent être appliquées aux niveau infrastructure et application d'un système Cloud (voir Section 6.3). Précisément, nous avons introduit des stratégies pour décrire l'élasticité horizontale et verticale, ainsi que pour les comportements de migration et de load balancing. Dans cette section, nous proposons une approche de simulation à base des files d'attente, comme expliqué précédemment, afin d'évaluer et de valider les comportements élastiques introduits. L'élas-ticité d'un système Cloud donné peut être simulée sous le spectre de l'élasticité horizontale ou l'élasticité verticale. Les actions de migration et de load balancing sont complémentaires à ces deux méthodes dimensionnements. Elles sont donc toujours appliquées lorsque leurs stratégies associées sont déclenchées. Dans cette section, nous étudions et évaluons les coûts et les performances du service Steam fonctionnant sur les stratégies d'élasticité définies. Afin de mener notre étude expérimentale, nous expliquons et paramétrons les simulations à conduire dans la sous-section suivante.

8.4.1 Paramétrage des simulations

Afin d'évaluer les stratégies d'élasticité définies, nous simulons l'exécution du service basé Cloud de la boutique en ligne Steam. Nous confrontons le système à un modèle de charge de travail imprévisible comme montré dans la Section 8.3.2. Ce modèle décrit une activité hautement variable du service durant une période de 200 unités de temps (t), selon trois phases principales :

— Phase 1 (t= $[0, 90]$) : Augmentation lente et progressive de la charge de travail en entrée.

— Phase 2 (t= $[91, 110]$) : Pic important de la charge de travail.

— Phase 3 (t= $[111, 200]$) : Baisse rapide de la charge de travail.

Cette activité représente les sollicitations de la boutique Steam suite à la publication d'offres promotionnelles de courte durée (*ventes flash*). Afin d'étudier le coût et les performances du service, nous configurons le système Cloud l'hébergeant selon le modèle de file d'attentes introduit, de la manière suivante :

— *Le taux d'arrivée* λ : Nous considérons que $\lambda = 500$. Cette valeur correspond au taux d'arrivée (des requêtes) maximal qui sera atteint durant la simulation. Néanmoins, les phases d'activités introduites précédemment décrivent l'évolution de la valeur de λ de 5% à 70% (phase 1), de 70% à 100% (phase 2) puis de 100% à 1% (phase 3).

— *Le taux de service* μ : Nous considérons que $\mu = 50$. Le taux de service indique le nombre moyen de requêtes traitées par chaque serveur (instance de service) à chaque unité de temps.

— *La taille de la file d'attente* Q : Nous considérons que $Q = \infty$. Comme expliqué dans la Section 8.3.2, quand la taille de la file d'attente est infinie, les requêtes ne sont pas rejetées en cas de saturation du système. Celles-ci restent en attente de service dans la file d'attente.

— *Le seuil maximal x de VMs* : Nous considérons que $x = 4$. Cela indique que le système peut approvisionner 4 machines virtuelles au maximum.
— *Le seuil maximal y d'instances de Service* : Nous considérons que $y = 4$. Cela indique que chaque machine virtuelle peut héberger un maximum de 4 instances du service de la boutique en ligne Steam au maximum.
— *Le seuil maximal z de requêtes* : Nous considérons que $z = 50$. Cela indique que chaque instance de service peut gérer un maximum de 50 requêtes à la fois (cf. Figure 8.2).

Afin d'étudier l'évolution du système en termes de ressources déployées, les simulations sont conduites à partir d'une configuration initiale du système où une instance du service de la boutique Steam est déployée au sein de l'unique machine virtuelle initialement en exécution. Aussi, les simulations sont conduites pour un seul serveur physique.

Principes de la simulation. Une fois le système paramétré, nous simulons son exécution afin d'étudier son élasticité. Précisément, nous simulons son comportement selon les stratégies d'élasticité horizontale et verticale. Ensuite, nous étudions les résultats obtenus afin d'évaluer les stratégies définies en termes de coûts et de performance. Notons que les résultats obtenus sont issus de notre outil de simulation et de monitoring introduit dans la Section 8.3.3. Les traces obtenues montrent l'évolution du système tout au long de la simulation de t=0 à t=200. À chaque unité de temps, le monitoring enregistre l'état du système en termes de requêtes reçues, de requêtes en attente, de nombre VMs déployées et de nombre d'instance de service en exécution. Durant les simulations menées, le monitoring montre l'approvisionnement des ressources ainsi que le nombre de requêtes en attente en réponse à la montée et à la baisse de la charge de travail. Lorsque la charge de travail augmente, le système approvisionne plus de ressources en déployant davantage d'instances de service et de machines virtuelles. En effet, la capacité d'hébergement d'une VM étant bornée, déployer des instances de service mènent à la surcharger. Cela amène au déploiement d'une autre VM et donc à la possibilité de déployer d'autres instances de service. Lorsque la charge de travail diminue, les instances de services sont inutilisées et doivent être supprimée. Cela amène à des VMs inutilisées qui doivent être libérées.

Validation des résultats. Afin de valider les résultats obtenus, nous comparons le nombre d'instances de service déployées avec les résultats donnés par la formule d'Erlang-C [Firdhous et al., 2011]. Cette formule donne le nombre minimal d'instances de service requises pour assurer un niveau de service donné, conformément à des valeurs données de λ et μ . Dans nos simulations, nous validons particulièrement les résultats obtenus durant la phase 2. En effet, durant cette phase le taux d'arrivée est stable ($\lambda = 500$) tandis qu'il varie dans les phases 1 et 3. De plus, la phase 2 décrit un pic important de charge. Nous considérons qu'il est plus pertinent d'étudier l'effort déployé en termes de ressources selon les différentes stratégies employées, pour un taux d'arrivée maximal et constant. Notons que le niveau de service à satisfaire considéré par la formule d'Erlang-C est donné par le nombre de clients qui sont servis dans la limite d'un temps jugé « acceptable ». Comme discuté dans la Section 8.3.2, cette notion est subjective et dépend de la nature du service en question. Dans notre étude, nous nous intéressons au service de la boutique en ligne de Steam et plus particulièrement aux commandes et achats en ligne effectués. En sachant qu'une transaction en ligne peut prendre quelques secondes afin d'être exécutée, nous fixons le délai « acceptable » pour la boutique Steam à une valeur arbitraire de 10 secondes. Pour nos validations, nous calculerons les résultats d'Erlang-C pour un niveau

de service à 100%, c'est-à-dire avec l'exigence que toutes les requêtes soient satisfaites dans la limite du délai acceptable. Notons que les valeurs de la formule d'Erlang-C sont calculées à l'aide de l'outil *CCOptim* disponible en ligne [Erlang, 2019].

8.4.2 Stratégies de dimensionnement Horizontal

Dans la Section 6.3, nous avons défini un nombre de stratégies d'élasticité horizontale multi-couche, opérant aux niveaux infrastructure et service d'un système Cloud. Les stratégies introduites définissent des comportements d'approvisionnement horizontal (*scale-out*) pour une *haute disponibilité* (H_Out1) et une *disponibilité limitée* (H_Out2) des ressources Cloud (VMs et/ou instances de service). Nous avons également défini une stratégie pour le désapprovisionnement horizontal (*scale-in*) s'appliquant en multi-couche lorsque le système est sur-dimensionné. Dans cette section, nous identifions quatre scénarios d'élasticité horizontale où les stratégies d'approvisionnement peuvent être composées au niveaux infrastructure et application de la manière suivante :

— *Scénario H1* : Application de la stratégie de haute disponibilité des ressources (H_Out1) aux niveaux infrastructure et application du système.

— *Scénario H2* : Application de la stratégie de disponibilité limitée des ressources (H_Out2) aux niveaux infrastructure et application du système.

— *Scénario H3* : Application de la stratégie de haute disponibilité (H_Out1) au niveau infrastructure et de la stratégie de disponibilité limitée (H_Out2) au niveau application.

— *Scénario H4* : Application de la stratégie de disponibilité limitée (H_Out2) au niveau infrastructure et de la stratégie de haute disponibilité (H_Out1) au niveau application.

Pour chaque scénario, nous simulons l'exécution du service de la boutique en ligne de Steam selon les configurations identifiées précédemment.

Scénario H1 : Haute disponibilité (H_Out1) aux niveaux infrastructure et application. Les traces décrivant l'exécution du système sont données dans la Figure 8.7. Durant la phase 1, le système s'adapte très rapidement de manière explosive passant de 1 instance de service et une 1 VM déployées à 16 instances de service et 4 VMs entre t=0 et t=30. En termes d'approvisionnement des ressources, le système maintient cette configuration durant la phase 2. Les ressources sont progressivement libérées durant la phase 3, lorsque la charge de travail diminue. Enfin, le système regagne sa configuration initiale (une VM et une instance de service) à t=175 lorsque la charge de travail diminue drastiquement.

Durant la phase 1, 15 requêtes en attente sont enregistrées en réponse à l'augmentation de la charge de travail. Le nombre moyen d'instances de service déployées est de 13,5 et le nombre moyen de VMs déployées est de 3,7. Enfin, le taux d'arrivée de requêtes moyen enregistré est de 201 requêtes par unité de temps. Durant la phase 2, le monitoring enregistre 16 instances de service et 4 VMs déployées, pour un taux d'arrivée moyen $\lambda = 500$. La phase 3 montre une moyenne de 6,7 instances de service et 2,1 VMs déployées pour une arrivée moyenne $\lambda = 129$. Notons que les phases 2 et 3 ne comptent aucune requête en attente de service. Enfin, les valeurs totales moyennes enregistrées sont de 15 requêtes en attente, 3,6 VM déployées, 13,3 instances de service en exécution et un taux d'arrivée moyen $\lambda = 210$. Ces résultats sont résumés dans le Tableau 8.1.

Du point de vue de la validation des résultats obtenus, nous notons des aspects positifs et négatifs de la manière suivante.

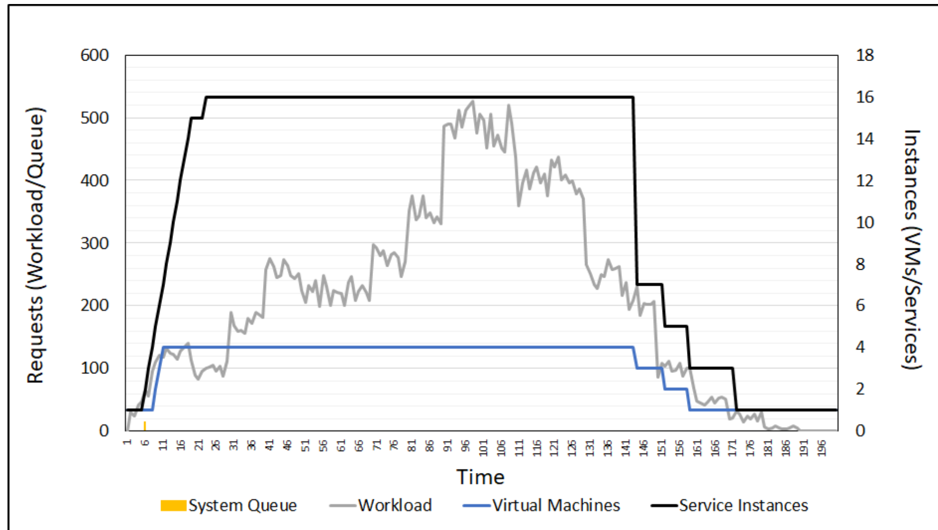


FIGURE 8.7 – Monitoring du scénario H1

TABLE 8.1 – Résultats du scénario H1

Métriques \Phases	Phase 1	Phase 2	Phase 3	Total
Requêtes en attente (%)	0,08	0	0	0,03
VMs déployées (moy.)	3,7	4	2,1	3,1
Services déployés (moy.)	13,5	16	6,7	11,3
Taux d'arrivée λ (moy.)	201	500	129	210

— *Aspects positifs* : Le monitoring enregistre un nombre négligeable de requêtes en attente durant une seule unité de temps, lorsque la charge de travail augmente. Le taux d'attente de service moyen assuré est de 0%.

— *Aspects négatifs* : Durant la phase 1, le monitoring montre que le système est sur-dimensionné. Celui-ci atteint rapidement sa capacité maximale en termes de ressources déployées avec 4 VMs et 16 instances de service ($x \times y$=16). Les ressources déployées sont les mêmes pour la phase 2 en dépit d'une charge de travail beaucoup moins importante. Cela est dû au fait que la stratégie employée (H_Out1) implique une haute disponibilité des ressources. Enfin, pour les mêmes valeurs de $\lambda = 500$ et $\mu = 50$, la formule d'Erlang-C indique que 14 instances de service sont requises au minimum pour assurer un niveau de service à 100%.

En vue des résultats obtenus, la stratégies H_Out1 aux niveaux infrastructure et service apporte des performances maximales. Cependant, elle induit des coûts importants en termes de ressources déployées et mène à un état de sur-dimensionnement considérable du système.

Scénario H2 : Disponibilité limitée (H_Out2) aux niveaux infrastructure et application. Les traces décrivant l'exécution du système sont données dans la Figure 8.8. Durant la phase 1, le système s'adapte progressivement à la montée de la charge de travail jusqu'à atteindre 7 instances de service et 2 VMs entre t=0 et t=75. Durant la phase 2, le système atteint un maximum de 12 instances de service et 4 VMs déployées. Les ressources sont progressivement libérées durant la phase 3, lorsque la charge de travail diminue, jusqu'au

retour à la configuration initiale en termes de ressources déployées.

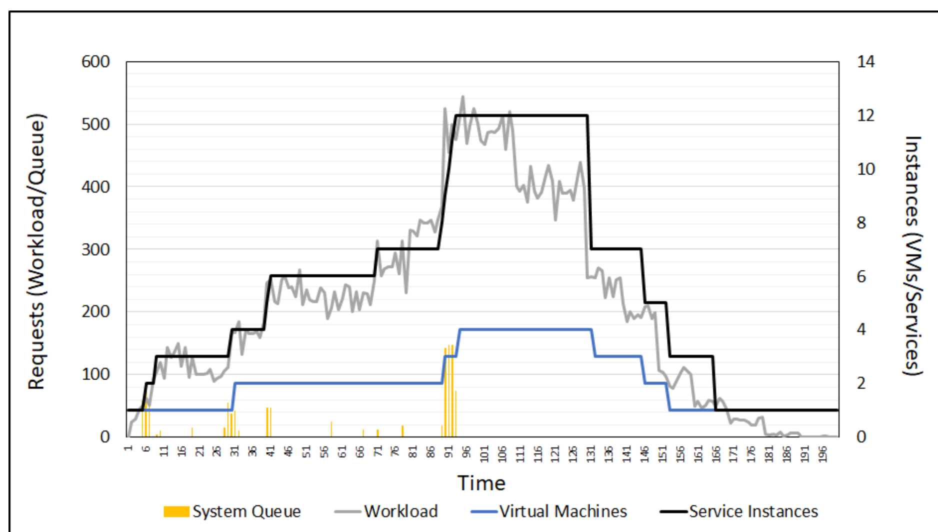


FIGURE 8.8 – Monitoring du scénario H2

Durant la phase 1, 569 requêtes en attente sont enregistrées en réponse à l'augmentation de la charge de travail. Le nombre moyen d'instances de service en exécution est de 4,8 et le nombre moyen de VMs déployées est de 1,6. Enfin, le taux d'arrivée de requêtes moyen enregistré est de 194 requêtes par unité de temps. Durant la phase 2, le monitoring enregistre 516 requêtes en attente lorsque le pic de charge a lieu. Une moyenne de 12 instances de service et 3,8 VMs déployées est enregistrée, pour un taux d'arrivée moyen $\lambda = 490$. La phase 3 montre une moyenne de 5 instances de service et 2 VMs déployées pour une arrivée moyenne $\lambda = 154$. Enfin, les valeurs totales moyennes enregistrées sont de 1085 requêtes en attente, 2 VM déployées, 5,6 instances de service en exécution et un taux d'arrivée moyen $\lambda = 207$. Ces résultats sont résumés dans le Tableau 8.2.

TABLE 8.2 – Résultats du scénario H2

Métriques \Phases	Phase 1	Phase 2	Phase 3	Total
Délai moyen d'attente (%)	12	18	0	15
VMs déployées (moy.)	1,6	4	2	2
Services déployés (moy.)	4,8	12	5	5,6
Taux d'arrivée λ (moy.)	194	490	154	207

Quant à la validation des résultats obtenus, nous notons des aspects positifs et négatifs de la manière suivante.

— *Aspects positifs* : Le monitoring montre que le système s'approvisionne en ressources (services et VMs) et absorbe sa charge de travail de manière globalement efficace. On peut observer que le système est rarement en état de surdimensionnement ou de sous-dimensionnement. En outre, la formule d'Erlang-C indique que 15 instances de service sont requises au minimum pour assurer un niveau de service à 100% pour les mêmes valeurs de $\lambda = 490$ et $\mu = 50$.

— *Aspects négatifs* : Durant la phase 1 et le début de la phase 2, le monitoring montre qu'un nombre assez important (1085) de requêtes en attente de service, lorsque la

charge de travail augmente. Cela est dû au fait que la stratégie H_Out2 implique une disponibilité limitée des ressources aux niveaux infrastructure et service.

La stratégies H_Out2 aux niveaux infrastructure et service apporte des coûts d'approvisionnement en ressources très réduits, avec des valeurs inférieures (et donc meilleures) que celles données par la formule d'Erlang-C. Cependant, elle implique de l'attente au niveau du service des requêtes avec un délai d'attente moyen de 15% (requêtes en attente/requêtes reçues).

Scénario H3 : H_Out1 au niveau infrastructure et H_Out2 au niveau application. Les traces décrivant l'exécution du système sont données dans la Figure 8.9. Durant la phase 1, le système s'adapte progressivement à la montée de la charge de travail jusqu'à atteindre 7 instances de service et 4 VMs entre t=0 et t=45. Durant la phase 2, le système atteint un maximum de 12 instances de service et 4 VMs déployées. Les ressources sont progressivement libérées durant la phase 3, lorsque la charge de travail diminue, jusqu'au retour à la configuration initiale en termes de ressources déployées.

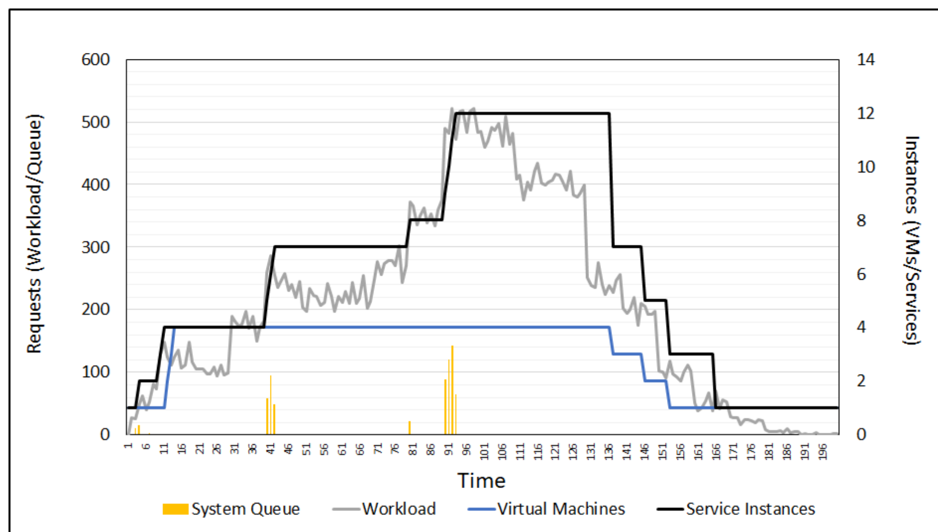


FIGURE 8.9 – Monitoring du scénario H3

Durant la phase 1, 253 requêtes en attente sont enregistrées en réponse à l'augmentation de la charge de travail. Le nombre moyen d'instances de service en exécution est de 5,5 et le nombre moyen de VMs déployées est de 3,5. Enfin, le taux d'arrivée de requêtes moyen enregistré est de 197 requêtes par unité de temps. Durant la phase 2, le monitoring enregistre 414 requêtes en attente lorsque le pic de charge a lieu. Un maximum de 12 instances de service et de 4 VMs déployées est enregistré, pour un taux d'arrivée moyen $\lambda = 486$. La phase 3 montre une moyenne de 5,3 instances de service et 3 VMs déployées pour une arrivée moyenne $\lambda = 155$. Enfin, les valeurs totales moyennes enregistrées sont de 670 requêtes en attente, 3 VM déployées, 7 instances de service en exécution et un taux d'arrivée moyen $\lambda = 208$. Ces résultats sont résumés dans le Tableau 8.3

Le scénario H3 montre la composition des deux stratégies précédemment étudiées. De ce fait, il présente des comportements similaires mais apportant des résultats quantitatifs distincts. Nous notons des aspects positifs et négatifs de la manière suivante.

— *Aspects positifs* : très similaire au scénario H2 au niveau application (du fait de l'application de la stratégie H_Out2), le scénario H3 montre que le système est globalement correctement dimensionné. De plus, il donne les mêmes résultats quant à

TABLE 8.3 – Résultats du scénario H3

Métriques \Phases	Phase 1	Phase 2	Phase 3	Total
Délai moyen d'attente (%)	6	12	0	8
VMs déployées (moy.)	3,5	4	3	3
Services déployés (moy.)	5,5	14	5,3	7
Taux d'arrivée λ (moy.)	197	486	155	208

la comparaison avec les valeurs données par la formule d'Erlang-C. Enfin, le système déploie rapidement des VMs et des instances de service au début de la phase 1, ce qui permet d'absorber la charge de travail de manière plus efficace, ce qui réduit le nombre de requêtes en attente de service.

— *Aspects négatifs* : Durant la phase 1, le système déploie rapidement des VMs jusqu'à atteindre sa capacité maximale ($x = 4$). Cependant, le nombre d'instances de service reste modéré, ce qui implique un taux d'utilisation très faible des VMs malgré les coûts importants liés à leur fonctionnement.

L'application de la stratégie H_Out1 au niveau infrastructure et de la stratégie H_Out2 au niveau application (scénario H3) apporte un compromis entre les scénarios H1 et H2. En vue des résultats obtenus, le système se montre légèrement plus performant avec un délai d'attente de 3% contre 5% enregistré pour le scénario H2. De plus, le système est légèrement plus économe avec un nombre moyen de 6 et 3 d'instances de service et de VMs respectivement déployées, contre 12,2 et 3,6 enregistrés pour le scénario H1.

Scénario H4 : H_Out2 au niveau infrastructure et H_Out1 au niveau application. Les traces décrivant l'exécution du système sont données dans la Figure 8.10. Durant la phase 1, le système s'adapte rapidement à la montée de la charge de travail jusqu'à atteindre 16 instances de service et 4 VMs entre t=0 et t=37. Durant la phase 2, le système atteint un maximum de 16 instances de service et 4 VMs déployées. Les ressources sont progressivement libérées durant la phase 3, lorsque la charge de travail diminue, jusqu'au retour à la configuration initiale.

Durant la phase 1, 680 requêtes en attente sont enregistrées en réponse à l'augmentation de la charge de travail. Le nombre moyen d'instances de service en exécution est de 13,5 et le nombre moyen de VMs déployées est de 3,6. Enfin, le taux d'arrivée de requêtes moyen enregistré est de 204 requêtes par unité de temps. Durant la phase 2, un maximum de 12 instances de service et de 4 VMs déployées est enregistré, pour un taux d'arrivée moyen $\lambda = 484$. La phase 3 montre une moyenne de 7.7 instances de service et 2.4 VMs déployées pour une arrivée moyenne $\lambda = 156$. Enfin, les valeurs totales moyennes enregistrées sont de 670 requêtes en attente, 2,6 VM déployées, 10,2 instances de service en exécution et un taux d'arrivée moyen $\lambda = 209$. Ces résultats sont résumés dans le Tableau.

Le scénario H4 présente des comportements similaires au scénario H3 mais décrit des comportements élastiques différents. Nous notons des aspects positifs et négatifs de la manière suivante.

— *Aspects positifs* : très similaire au scénario H1 au niveau application (du fait de l'application de la stratégie H_Out1), le scénario H4 montre que le système s'adapte rapidement en vue d'absorber sa charge de travail. Ainsi les performances enre-

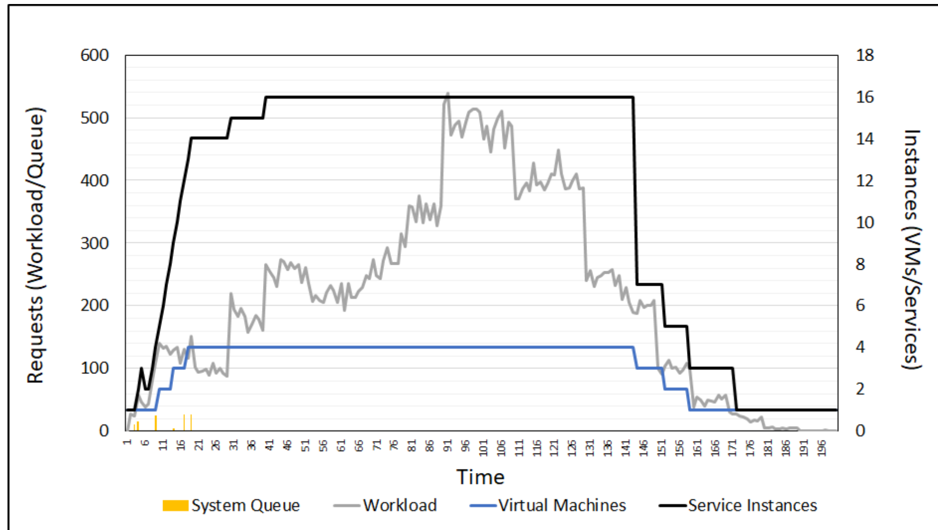


FIGURE 8.10 – Monitoring du scénario H4

TABLE 8.4 – Résultats du scénario H4

Métriques\Phases	Phase 1	Phase 2	Phase 3	Total
Délai moyen d'attente (%)	9	0	0	2
VMs déployées (moy.)	3,1	4	2,4	2,6
Services déployés (moy.)	12,5	16	7,7	10,2
Taux d'arrivée λ (moy.)	204	484	156	209

gistrées sont très bonnes avec 2% de requêtes en attente. De plus, les VMs ne sont pas déployées aussi rapidement que pour le scénario H1, ce qui induit des coûts légèrement moins importants.

— *Aspects négatifs* : Durant la phase 1, le système déploie rapidement sa capacité maximale en termes de VMs et instances de service. Cela conduit à un état général de sur-dimensionnement. De plus, le système donne les mêmes résultats que le scénario H1 quant à la comparaison avec les résultats donnés par Erlang-C.

L'application de la stratégie H_Out2 au niveau infrastructure et de la stratégie H_Out1 au niveau application (scénario H4) apporte un meilleur compromis que H3 entre les scénarios H1 et H2. En vue des résultats obtenus, le système se montre aussi performant que pour le scénario H1. De plus, le monitoring montre, notamment pour la phase 1, que les VMs sont déployées moins rapidement que pour les scénarios H1 et H3.

Bilan. Dans cette sous-section, nous avons simulé le comportement élastique du service basé Cloud de la boutique en ligne Steam. Nous avons proposé quatre scénarios d'exécution en appliquant les deux stratégies d'élasticité Horizontale H_Out1 et H_Out2 en multi-couches « *cross-layer* », c'est-à-dire aux niveaux infrastructure et application du système Cloud simulé. Il est intéressant de remarquer que les différentes combinaisons décrivent des comportements (politiques) de haut-niveau différents, comme montré dans la Figure 8.11.

Intuitivement, en appliquant la stratégie H_Out1 pour la haute disponibilité des ressources aux niveau infrastructure et application (scénario H1), nous obtenons une poli-

tique de **hautes performances** (*high performance*) avec un taux d'attente minime (0,1%). Cependant, cela implique des coûts de déploiement importants avec 78% de la capacité maximale en VMs et 70% en instances de service déployées. Aussi, le système présente une efficience faible avec un taux d'utilisation des instances de service (*system load*) de 37% en moyenne.

Réciproquement, l'application de la stratégie H_Out2 pour la disponibilité limitée des deux ressources en multi-couches (scénario H2) décrit une politique de **hautes économies** (*high economy*) avec 52% des VMs et 35% des instances de service déployées. Cependant, cela implique des performances relativement faibles avec un taux d'attente de service de 15%. Néanmoins, le système montre une efficience considérable avec un taux d'utilisation des instances de service de 73% en moyenne.

La combinaison H_Out1 et H_Out2 aux niveaux infrastructure et application (scénario H3) apporte un compromis en termes de coûts et de performances, tout en avantageant la disponibilité des ressources au niveau infrastructure (VMs). Cette combinaison décrit une politique de ***haute disponibilité de l'infrastructure*** (*high infrastructure availability*) où le système est en mesure de rapidement s'approvisionner en VMs aux prémisses d'une augmentation soudaine de la charge de travail. En effet, dans notre simulation, ce scénario montre de bonnes performances, avec un délai d'attente de 8% (contre 0.1% pour H1). Cependant, même si l'efficience achevée est similaire (mais inférieure) à H2, avec un taux d'utilisation des services de 68%, H3 présente des coûts d'infrastructure plus importants avec 74% des VMs déployées contre 52% pour H2.

La combinaison H_Out1 et H_Out2 aux niveaux application et infrastructure (scénario H4) montre un comportement légèrement subtil. Cette combinaison décrit une politique d'***optimisation des coûts d'infrastructure*** (*infrastructure costs optimization*). Si l'élasticité au niveau logiciel reste similaire (mais inférieure) à H1 avec 63% des services déployés contre 70%, ce scénario montre des coûts d'infrastructure légèrement inférieurs avec 65% des VMs déployées contre 78%. De plus, le système y achève une meilleure efficience avec un taux d'utilisation des services de 45% contre 37% pour H1. Enfin, des performances légèrement inférieures mais équivalentes sont enregistrées avec 3% de délai d'attente contre 0% pour H1.

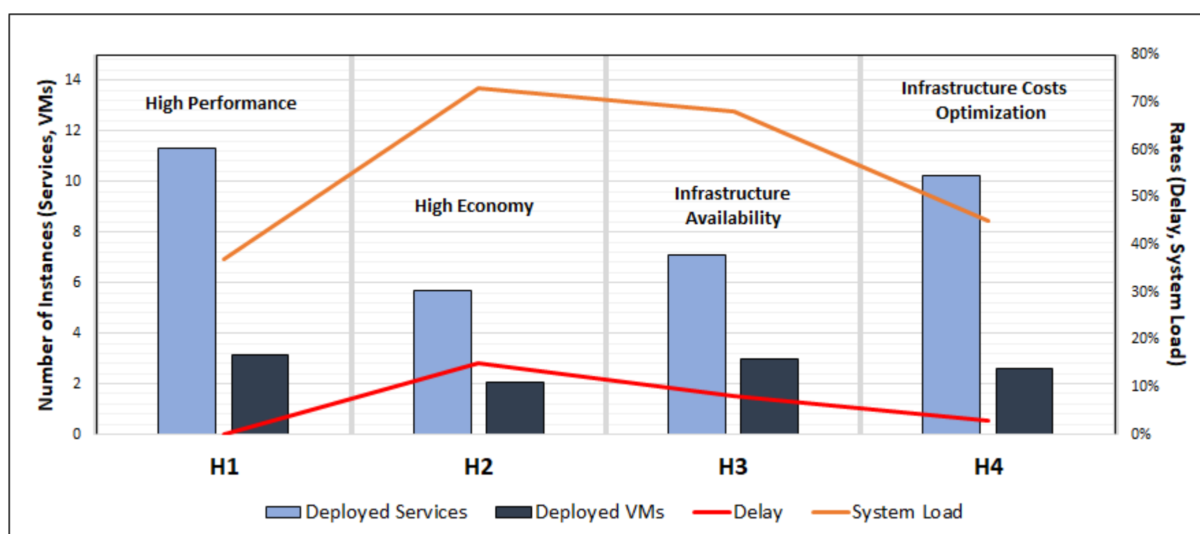


FIGURE 8.11 – Évaluation de l'élasticité horizontale en multi-couches

En conclusion, nous observons que l'état du déploiement des ressources influe considé-

rablement les performances, les coûts, ainsi que l'efficience du système Cloud étudié. En effet, limiter le déploiement en termes d'instance de service mène à une meilleure efficience. Cependant, cela implique également de moins bonnes performances pour des coûts réduits. Réciproquement, une haute disponibilité au niveau application, implique globalement un déploiement d'infrastructure plus important. Cela se traduit par des coûts plus importants et une efficience relativement faible. Cependant, cela assure de bonnes performances.

8.4.3 Migration et Load Balancing

Les graphes montrés ne permettent pas d'observer les mécanismes de load balancing et de migration aux niveaux application et infrastructure respectivement (voir Section 6.3). Cela est néanmoins visible au niveau des traces produites par notre outil de monitoring. Comme le montre la Figure 8.12, l'outil enregistre, pour chaque instant t de la simulation, les différentes opérations exécutées en fonction de l'état observé du système. La trace montre l'activité autonomique du système cloud à l'instant t=93 (début de la phase 2 du scénario H2) où le système distribue les requêtes en entrée entre les 11 différentes instances de service déployées. Ensuite, un service est migré d'une VM à une autre. Enfin, une nouvelle instance de service est déployée afin de prendre en charge les 66 requêtes en attente de service dans la file du système. Nous observons également que le load balancing transfert 50 requêtes à chaque instance de service. Cela vient du fait que le seuil d'hébergement maximal en termes de requêtes soit fixé à z=50. Cela indique également que toutes les instances de service sont surchargées, ce qui justifie le déploiement d'une nouvelle instance.

```
tick = 93
Incoming workload = 502
System Queue length = 616 transferring 50 requests to S@15500897331550750108083 NEW System Queue length = 566
System Queue length = 566 transferring 50 requests to S@14424071701550750108093 NEW System Queue length = 516
System Queue length = 516 transferring 50 requests to S@10285661211550750108093 NEW System Queue length = 466
System Queue length = 466 transferring 50 requests to S@11181408191550750108103 NEW System Queue length = 416
System Queue length = 416 transferring 50 requests to S@18082530121550750108108 NEW System Queue length = 366
System Queue length = 366 transferring 50 requests to S@5894319691550750108108 NEW System Queue length = 316
System Queue length = 316 transferring 50 requests to S@12521699111550750108108 NEW System Queue length = 266
System Queue length = 266 transferring 50 requests to S@6853251041550750108123 NEW System Queue length = 216
System Queue length = 216 transferring 50 requests to S@4601419581550750108123 NEW System Queue length = 166
System Queue length = 166 transferring 50 requests to S@11631578841550750108123 NEW System Queue length = 116
System Queue length = 116 transferring 50 requests to S@3565735971550750108123 NEW System Queue length = 66
Service instance S@6853251041550750108123 migrated from V@2101973421$$1550750108108 to V@1956725890$$1550750108123 TOTAL VMs = 4/ Services =11
Service instance S@1735600054155075010812 3 deployed to V@1956725890$$1550750108123
```

FIGURE 8.12 – Trace du monitoring de H2 à t=93

8.4.4 Stratégies de dimensionnement Vertical

Dans la Section 6.3, nous avons défini deux stratégies d'élasticité verticale pour le redimensionnement du système Cloud au niveau infrastructure. Précisément, cela consiste en une stratégie (V_Up) pour l'ajout, et une stratégie (V_Down) pour le retrait des ressources informatiques (CPU et RAM) aux VMs déployées. L'élasticité verticale introduit des comportements différents de deux de l'élasticité horizontale. L'élasticité verticale consiste à redimensionner les entités existantes (VMs, services) de manière à augmenter leur capacité (en leur allouant plus de ressources), tandis que l'horizontale duplique les entités en gardant la même configuration initiale.

Principes de la simulation de l'élasticité verticale. Du point de vue de l'élasticité verticale, le nombre d'entités déployées (VMs, services) n'est pas pertinent car celui-ci ne

changera pas au long de la simulation. D'un autre côté, il est intéressant d'observer que le comportement vertical peut être pris en charge par le modèle à base des files d'attentes utilisé. En effet, ajouter/retirer de la mémoire RAM à une VM peut résulter en l'augmentation/diminution du seuil d'hébergement maximal des requêtes pour les instances de service déployées au sein de cette VM. De la même manière, ajouter/retirer des cœurs de CPU à cette VM augmentera/diminuera la capacité des services qui y sont déployés en termes de performances. En d'autres termes, ajouter ou retirer de la RAM a pour effet d'augmenter ou de baisser la valeur du seuil z, représentant la taille de la file d'attente au niveau des services. D'un autre côté, ajouter ou retirer des cœurs de CPU a pour effet d'augmenter ou de baisser la valeur du taux de service μ , qui indique la performance des instances de service déployées. De ce fait, nous conduisons une simulation de l'exécution du service de la boutique Steam à partir du même état initial (1 VM et 1 instance de service) et observons l'évolution de la capacité du système en termes de requêtes pouvant être prises en charge à chaque instant. Nous considérons que la capacité est donnée par $c = s \times \mu$ où s est le nombre d'instances de service et μ est le taux de service moyen. Étant donné que s=1, la capacité observée représentera l'évolution du taux de service en résultat à l'ajout et retrait de ressources comme expliqué précédemment. Quant à la taille de la file d'attente de l'instance du service, l'augmentation de sa capacité permettra de prendre en charge plus de requêtes lorsque la charge de travail augmente. Cela permettra de minimiser le taux de requêtes en attente de service dans la file d'attente du système.

Validation des résultats. Du point de vue quantitatif, le recours à la formule d'Erlang-C est cette fois inadéquat. En effet, la formule d'Erlang-C calcule le nombre d'instances de service minimal pour assurer un certain niveau de service à partir d'un taux d'arrivée λ et d'un taux de service μ moyens stables. Or dans le cas de l'élasticité verticale, le nombre d'instances de service est fixe mais le taux de service est variable. En outre, il est difficile de déterminer de manière générique quelle serait une allocation de ressources (CPU, RAM) suffisante afin de satisfaire efficacement une charge de travail donnée. Nous considérons que l'approvisionnement initial du système étudié est caractérisé par une unité de RAM et une unité de CPU déployées. Cela résulte en $z = 50$ et $\mu = 50$ initialement. En d'autres termes, une unité de CPU correspond à une capacité de 50 requêtes traitées et une unité de RAM correspond à 50 requêtes dans la file d'attente du service. De ce fait, nous raisonnons d'une manière arbitraire, de façon à simplement augmenter/diminuer la capacité de z de 50 à l'ajout/retrait d'une unité de RAM. La capacité de μ est également augmentée/diminuée de 50 à l'ajout/retrait d'une unité de CPU. Finalement, nous évaluerons les comportements obtenus en analysant les états de surdimensionnement et de sous-dimensionnement du système compte tenu des ressources déployées.

Scénario V : Application des stratégies V_Up et V_Down. Le résultat du monitoring du système est donné par la Figure 8.13. Durant la phase 1, le système s'adapte à la montée de la charge de travail en augmentant sa capacité (c) en termes de requêtes prises en charge ainsi que la taille de la file d'attente du service (q) (en allouant des unités de CPU et de RAM). Un maximum d'une capacité et d'une taille de la file de 650 requêtes est atteint lors de la phase 2. À l'instar des scénarios observés pour l'élasticité horizontale, le système diminue progressivement sa capacité et la taille de la file, durant la phase 3, jusqu'à atteindre sa configuration initiale.

Durant la phase 1, 480 requêtes en attente sont enregistrées en réponse à l'augmen-

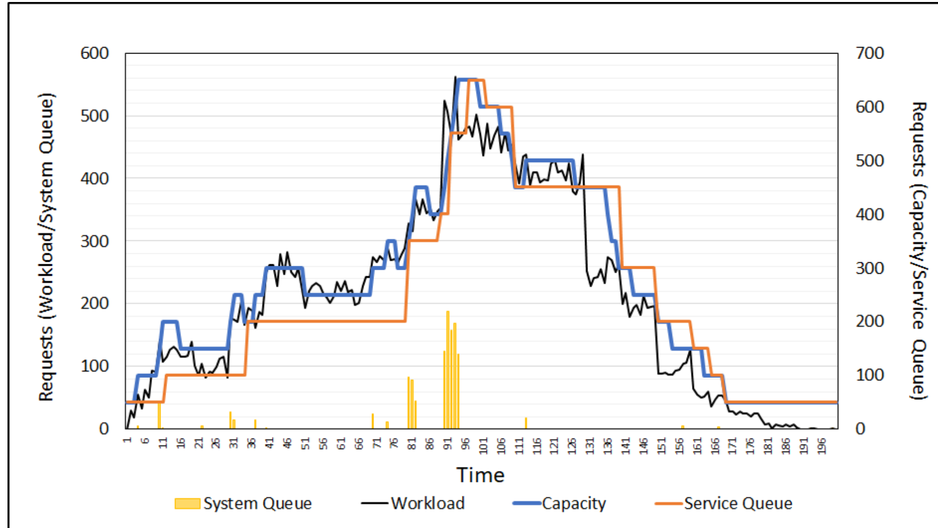


FIGURE 8.13 – Monitoring de l'élasticité verticale (scénario V)

tation de la charge de travail. La capacité moyenne du système est enregistrée à $c = 245$ requêtes et la taille moyenne de la file d'attente à $q = 175$. Le taux d'arrivée moyen est $\lambda = 201$. Durant la phase 2, un total de 757 requêtes en attente de service est enregistré. La capacité moyenne du système est de c=580 requêtes et la taille moyenne de la file d'attente est de $q = 573$, pour un taux d'arrivée $\lambda = 473$. Durant la phase 3, le monitoring enregistre 28 requêtes en attente de service. Les valeurs moyennes observées sont $c = 226$, $q = 231$ et $\lambda = 155$. Enfin, les valeurs totales moyennes enregistrées pour cette simulation sont de 1141 requêtes en attente, une capacité moyenne $c = 271$, une taille de la file d'attente $q = 241$ et un taux d'arrivée $\lambda = 208$. Ces résultats sont résumés dans le Tableau 8.5. Comme expliqué précédemment, il est possible de déduire le nombre d'unités de CPU et de RAM déployées à partir de la capacité du système (c) et de la taille de la file d'attente au niveau service (q) respectivement. Ainsi $RAM = q/50$ et $CPU = c/50$.

TABLE 8.5 – Résultats du scénario V

Métriques \Phases	Phase 1	Phase 2	Phase 3	Total
Délai moyen d'attente (%)	4,7	8,6	0,6	6,4
Unités ce CPU (moy.)	4,9	11,6	4,5	5,42
Unités de RAM (moy.)	3,5	11,47	4,6	4,8
Taux d'arrivée λ (moy.)	201	473	155	208

En analysant les comportements obtenus, nous notons des aspects positifs et négatifs de la manière suivante.

— Aspects positifs : Le monitoring montre que le système s'adapte de manière très souple à sa charge de travail aussi bien à la montée qu'à la baisse. La capacité enregistrée épouse globalement la charge de travail en entrée.

— Aspects négatifs : Un taux d'attente assez important est enregistré en début de la phase 2 (pic de charge).

Le scénario V montre que les stratégies d'élasticité verticale V_Up et V_Down décrivent un comportement élastique très efficace avec un taux limité de ressources allouées. Glo-

balement, le système montre un comportement de dimensionnement très proche de la demande réelle. Ainsi, nos stratégies d'élasticité verticale décrivent une politique de haut niveau pour une haute efficience des ressources (*high resources efficiency*).

8.4.5 Comparaison de l'élasticité Horizontale et Verticale

Dans cette sous-section, nous procédons à une comparaison des résultats obtenus pour l'élasticité horizontale et verticale. Pour ce faire, nous commençons par calculer le nombre de ressources (CPU, RAM) déployées ainsi que leur taux d'utilisation pour le dimensionnement horizontal (scénarios H1 − H4). Dans le cas de l'élasticité verticale, nous considérons qu'une VM dispose d'une unité de CPU et d'une unité de RAM. Nous avons supposé qu'une unité de CPU correspondait à une capacité de traitement de 50 requêtes, et qu'une unité de RAM correspondait à une capacité de 50 requêtes dans la file d'attente du service déployé.

En supposant qu'une instance de service requière une unité de CPU et une unité de RAM pour pouvoir être exécutée, nous pouvons déduire qu'une VM, dans le cas des scénarios d'élasticité horizontale, dispose de 4 unités de CPU et de 4 unités de RAM. Nous avions fixé le seuil d'hébergement maximal pour chaque VM en termes d'instances de service à $y = 4$. C'est-à-dire qu'une VM peut héberger jusqu'à 4 instances de service en exécution, d'où la nécessité de disposer des ressources nécessaires à cet effet. Ainsi, le nombre d'unités de ressources déployées (*deployed*) est donné par $CPU_D = RAM_D = 4 \times VMs$. Où VMs est le nombre de machines virtuelles déployées. D'un autre côté, le nombre d'unités de ressources réellement utilisées (*used*) est donné par $CPU_U = RAM_U = S$, où S est le nombre d'instances de service déployées. Dans le cas de l'élasticité verticale, les ressources allouées sont toutes utilisées, du fait qu'il n'y ait qu'une seule instance de service déployée. En d'autres termes $CPU_D = CPU_UetRAM_D = RAM_U$.

la Figure 8.14 montre un comparatif de la consommation des ressources ainsi que les coûts et performances liées à l'élasticité horizontale et verticale. Les valeurs affichées dans le graphe sont explicitées dans le Tableau 8.6. Elles sont calculées de la manière décrite à partir des résultats des simulations discutés précédemment dans les sous Sections 8.4.2 et 8.4.4.

Avantages de l'élasticité verticale. Les résultats montrent que l'élasticité verticale donne des résultats bien supérieurs à ceux obtenus pour l'élasticité horizontale. En effet, les comportements élastiques montrent une bonne efficience avec un taux d'utilisation du service à 79%. Le dimensionnement vertical résulte également en de bonnes performances avec un taux d'attente de 7%. Ainsi le scénario V apporte des résultats similaires au scénario H2 en termes d'efficience des ressources tout en garantissant un niveau de performances proche de H3. Cependant, l'élasticité verticale se distingue réellement de l'horizontale en offrant un état de dimensionnement plus optimisé. Les ressources déployées (CPU, RAM) sont en effet utilisées de manière plus efficace. Les différents scénarios d'élasticité horizontale montrent que les ressource sont globalement sous-utilisées.

Avantages de l'élasticité horizontale. L'élasticité verticale semble en tout point supérieure à l'élasticité horizontale, notamment du point de vue de l'efficience des ressources et des coûts liés à leur déploiement. Néanmoins, l'avantage de l'élasticité horizontale réside en deux autres notions liées au fonctionnement même du système. Si l'élasticité verticale promeut l'économie et l'efficience, l'élasticité horizontale privilégie la fiabilité et la dis-

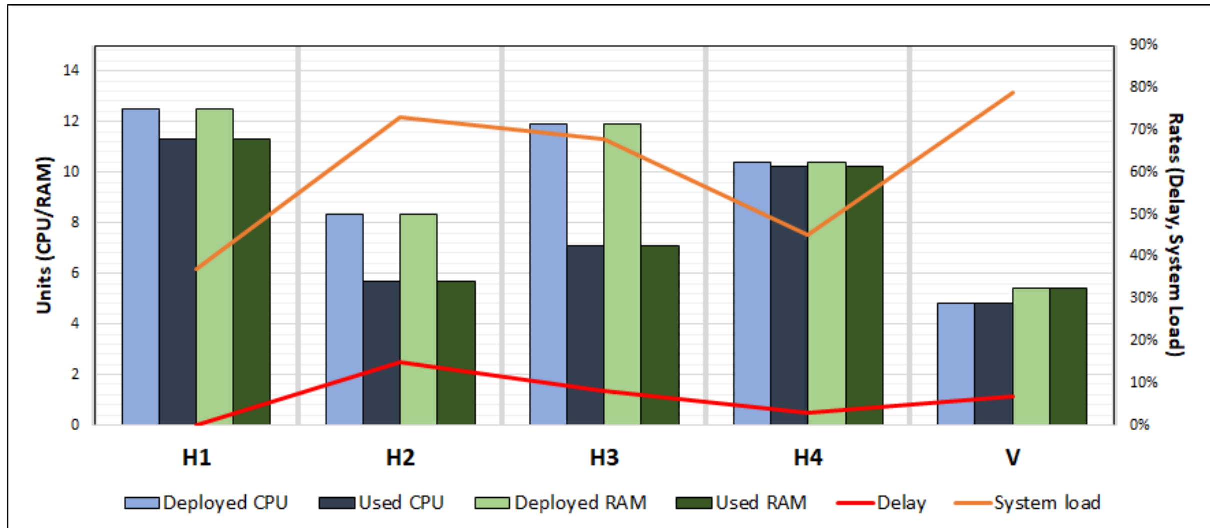


FIGURE 8.14 – Comparaison de l'élasticité horizontale et verticale

TABLE 8.6 – Comparaison de l'élasticité horizontale et verticale

Résultats \Scénarios	H1	H2	H3	H4	V
Unités CPU déployées	12,52	8,32	11,92	10,4	4,8
Unités CPU utilisées	11,32	5,67	7,1	10,2	4,8
Unités RAM déployées	12,52	8,32	11,92	10,4	5,42
Unités RAM utilisées	11,32	5,67	7,1	10,2	5,42
Taux d'attente	0%	15%	8%	3%	7%
Taux d'utilisation des services	37%	73%	68%	45%	79%

ponibilité du système. En effet, disposer de plusieurs nœuds déployés (VMs et instances de service) donne accès à un déploiement distribué du service exécuté (boutique en ligne Steam dans notre étude). Un tel déploiement résulte en une fiabilité accrue du système lui permettant de maintenir un bon fonctionnement en cas de pannes matérielles ou logicielles (e.g., arrêt d'une instance de service, panne de réseau, arrêt de fonctionnement d'une VM). De plus, la disponibilité du service est assurée même si une panne au niveau d'un nœud entraine une baisse temporaire des performances. D'un autre côté, l'élasticité verticale permet un déploiement de type centralisé. Cela décrit une architecture de type SPOF (*single point of failure*) ou point unique de défaillance, où l'arrêt (e.g. panne, mise à jour) d'un nœud (VM, instance de service) résulte en l'interruption totale et potentiellement prolongée du service [Suleiman et al., 2012].

8.5 Conclusion et discussion

Dans ce Chapitre, nous avons proposé une étude expérimentale des stratégies d'élasticité introduites, afin de valider les comportements obtenus d'un point de vue quantitatif. Nous nous sommes intéressés à l'étude du service basé Cloud de la boutique en ligne Steam. Dans un premier temps, nous avons appliqué notre approche de modélisation formelle afin de représenter les configurations du service Cloud Steam. Ensuite, nous avons abordé les possibilités de vérification formelle de l'élasticité pour les configurations obtenues. Enfin,

nous avons proposé une approche à base de files d'attente afin de conduire des simulations du fonctionnement du service étudié selon les différentes stratégies d'élasticité introduites dans ce manuscrit. À ces fins, nous avons conçu un outil pour la simulation et le monitoring des comportements élastiques autonomiques d'un système Cloud à base de files d'attente.

Dans notre étude expérimentale, nous avons simulé l'exécution du service de la boutique Steam via un modèle de variation imprévisible de la charge de travail en entrée (workload), de manière à décrire une sollicitation croissante, un pic de charge, puis une baisse rapide de celle-ci. Pour la même configuration initiale du système étudié, en termes de ressources déployées (VMs, instances de service, CPU, RAM), nous avons simulé son fonctionnement selon l'élasticité horizontale puis l'élasticité verticale.

Pour la simulation de l'élasticité horizontale, nous avons identifié 4 scénarios d'exécution (H1, H2, H3 et H4). Ces scénarios résultent de la composition multi-couches des deux stratégies de *scale-out* H_Out1 et H_Out2, au niveaux infrastructure et application du système. Nous avons montré que les différentes combinaisons décrivaient plusieurs politiques de haut niveau, à savoir :

— Scénario H1 : *hautes performances (high performance)*

— Scénario H2 : *hautes économies (high economy)*

— Scénario H3 : *optimisation des coûts d'infrastructure (infrastructure costs optimization)*

— Scénario H4 : *haute disponibilité de l'infrastructure (high infrastructure availability)*

Quant à la simulation de l'élasticité verticale, nous avons montré l'application des stratégies de *scale-up* et de *scale-down* (V_Up et V_Down) sur le système à travers le scénario V. En résultat, nous avons montré, à travers le scénario V que les stratégies d'élasticité verticale décrivent la politique de haut niveau suivante :

— Scénario V : *haute efficience des ressources (high resources efficiency)*

Enfin, nous avons mené une comparaison afin d'analyser les avantages et inconvénients liés à l'élasticité horizontale et verticale en termes de ressources déployées, d'efficience et de performance. Á travers ce dernier Chapitre dédié à nos contributions de thèse, nous aurons répondu à notre quatrième et dernier objectif de recherche (OR4) visant à évaluer les performances et les coûts liés aux comportements du contrôleur d'élasticité définis.

Discussion. L'étude expérimentale menée montre le comportement élastique du système Cloud simulé. Les résultats obtenus pour l'élasticité horizontale sont globalement en adéquation avec ceux obtenus via la formule d'Erlang-C. Á travers les simulations menées, nous avons montré que l'élasticité horizontale mettait l'accent sur la disponibilité et la fiabilité du système, là où l'élasticité verticale était plus axée sur l'efficience des ressources. En vue des différents comportements élastiques produits, nous pouvons nous demander : « quel scénario apporte les meilleurs résultats ? » ou encore « quelle est la meilleure stratégie à adopter ? ». Pour y répondre, revenons sur le concept de « bonne » stratégie. Nous estimons que cette notion reste très subjective et est très difficilement formalisable. En même temps, les mesures de performances (taux d'attente) et de coûts (ressources déployées) sont des notions objectives. Il est donc légitime de considérer qu'une bonne stratégie serait celle qui amènerait des performances maximisées pour des coûts minimisés. Néanmoins, les résultats dépendent fortement du cas d'utilisation et de l'activité rencontrée (charge de travail).

Notre approche de modélisation apporte la notion de choix et de liberté dans le paramétrage des comportements élastiques d'un système Cloud. Nous fournissons un moyen de s'adapter en fonction du contexte particulier et des restrictions et contraintes du système, notamment en termes de budget, d'intensité de la charge de travail ou encore de la nature du service exécuté. Ainsi, notre solution pour la gestion de l'élasticité permet au fournisseur d'un service Cloud de paramétrer les comportements élastiques de son service en fonction de ses exigences et contraintes. Du point de vue de la modélisation proposée, beaucoup de concepts sont à prendre en compte. Les seuils d'hébergement maximaux (x, y, z) peuvent être adaptés de manière à répondre à certaines considérations liées au système. Le seuil maximal x de VMs peut modéliser les contraintes financières et le budget du fournisseur de service. Plus le budget est important, plus le fournisseur pourra déployer des VMs pour héberger son service. Le seuil y d'instances de service peut indiquer le profil matériel ($offering$) des VMs à déployées. Un VM disposant de plus de ressources (CPU, RAM) peut accueillir un plus grand nombre de services en exécution. Comme abordé précédemment, le choix de taille Q de la file d'attente, ainsi que du taux de service μ, peut être adapté à la nature du service à exécuter.

Enfin, notre approche de modélisation des comportements élastiques peut être appliquée en vue d'établir des contrats d'accords de niveau de service (*Service Level Agreements SLA*) [Patel et al., 2009], entre le fournisseur du service et le fournisseur de l'infrastructure Cloud. En effet, par le biais de la simulation des comportements définis, nous avons montré qu'un système peut s'adapter de manière correcte à des scénarios de charge de travail imprévisible. À travers notre outil de simulation, un fournisseur de service Cloud peut simuler son système en le confrontant à une infinité des situations. Cela lui permettrait d'avoir des estimations des ressources requises afin de satisfaire un certain niveau de service, en plus des coûts et performances liées au fonctionnement de son système. En variant les possibilités d'adaptations élastiques, à travers les différentes stratégies introduites, le fournisseur de service peut opter pour les comportements qui correspondent le mieux à ses besoins et exigences, et qui satisfont au mieux ses contraintes.

Chapitre 9

Conclusion Générale

Sommaire

9.1 Contributions 149
9.2 Perspectives 150

L'élasticité est un concept clé du Cloud Computing. Les systèmes Cloud dits élastiques se caractérisent par leur capacité d'adapter, de manière autonomique, la consommation des ressources informatiques afin de maintenir une bonne qualité de service (QoS), tout en minimisant les coûts de fonctionnement. Dans les environnements Cloud, de nombreuses solutions académiques et commerciales ont été proposées pour automatiser la gestion et la planification de deux types d'élasticité : horizontale et verticale. La mise en œuvre de l'élasticité dans un système Cloud est basée sur deux types de contrôleurs d'élasticité : réactifs et proactifs, adoptant une boucle de contrôle autonomique MAPE-K (Monitor, Analyse, Plan, Execute - Knowledge). Néanmoins, toutes les solutions existantes actuellement, proposées par des grands acteurs du marché Cloud ou bien venant du monde de la recherche académique, sont encore immatures. En effet, d'une part, les solutions commerciales présentent beaucoup de limites et de contraintes pour leurs utilisateurs et sont généralement très spécifiques à un type donné de systèmes, ou fonctionnelles pour un fournisseur Cloud en particulier. D'une autre part, certaines propositions du monde académique se montrent trop conceptuelles, théoriques ou abordant le problème de l'élasticité à un haut niveau d'abstraction. Ces approches sont parfois très coûteuses, difficiles à mettre en œuvre voire inapplicables dans les environnements Cloud.

Afin de fournir une bonne gestion de l'élasticité, il est indispensable de s'appuyer sur un modèle qui permet de décrire les architectures des systèmes élastiques basés Cloud, ainsi que leur comportement. La modélisation est une tâche impérative qui permet de déterminer et comprendre les changements structurels et les dépendances comportementales dans un système élastique basé Cloud afin d'éviter l'émergence des comportements provoquant des situations indésirables telles que l'instabilité dans l'allocation des ressources et la dégradation de la qualité de service. Dans ce contexte, les méthodes formelles caractérisées par leur efficacité, fiabilité et précision, présentent une solution efficace pour relever ce défi et pour éliminer les ambiguïtés sémantiques et la complexité provenant de la tâche de modélisation. Ceci permet de raisonner rigoureusement sur les spécifications formelles obtenues pour démontrer leur validité à travers plusieurs techniques de simulations et vérifications formelles.

La modélisation et l'analyse formelle des systèmes Cloud sont des problématiques d'ac-

tualités qui ont attiré l'attention du milieu académique et abouti à plusieurs approches en se basant sur des formalismes différents. Cependant, à cause de leur complexité, peu de travaux ont abordé les aspects comportementaux en termes d'élasticité et d'approvisionnement autonomique dans le Cloud. Quelques tentatives de recherche récentes ont orienté leurs efforts vers la modélisation et la vérification formelle des aspects comportementaux relatifs aux systèmes élastiques basés Cloud. Lors de notre étude des travaux existants dans ce domaine, nous avons soulevé l'absence d'une approche générique et complète pour modéliser et analyser ces systèmes. Ces approches formelles se concentrent essentiellement sur le niveau infrastructure ou application du modèle Cloud et l'élasticité horizontale. En outre, la majorité de ces approches proposent des solutions simples et limitées puisqu'elles ignorent les aspects architecturaux ainsi que les dépendances structurelles et comportementales internes dans les systèmes élastiques basés Cloud. De plus, les travaux cités ne couvrent qu'une partie du cycle de vie des approches qu'elles proposent, à savoir : la conception, l'implémentation, la vérification qualitative et la validation quantitative.

9.1 Contributions

Dans cette thèse, nous avons proposé une solution complète à fondements formels pour la spécification des systèmes Cloud, la vérification qualitative et la validation quantitative de leurs comportements élastiques. Notre approche de modélisation repose sur l'association complémentaire du formalisme des systèmes réactifs bigraphiques et du langage de spécification formelle Maude. D'un autre côté, nous nous inspirons de la théorie des files d'attente pour fournir un support robuste pour l'évaluation quantitative et la simulation des comportements définis. Dans ce manuscrit, nous avons présenté les contributions suivantes.

1. **Sémantique bigraphique structurelle.** Nous avons adopté les bigraphes et leur discipline de typage comme cadre formel pour la spécification des aspects structurels des systèmes Cloud élastiques. Ce modèle générique et modulaire permet d'exprimer les éléments pertinents d'une architecture Cloud en multi-couche, c'est à dire aux niveaux infrastructure et application. La logique de typage associée aux bigraphes conçus permet de décrire une sémantique robuste et détaillées pour les systèmes Cloud et les entités les composant (serveurs physiques, machines virtuelles, instances de service Cloud, requêtes, ressources informatiques déployées).
2. **Sémantique BRS comportementale et stratégies d'élasticité multi-couches.** Nous avons proposé une sémantique basée sur les systèmes réactifs bigraphiques (BRS) pour la modélisation des actions de reconfiguration dynamiques des systèmes Cloud. Les reconfigurations du système sont modélisées en termes de règles de réaction bigraphiques s'appliquant au niveau des différentes ressources matérielles et logicielles déployées au sein du système (multi-couches). À partir des différentes actions identifiées, nous avons défini plusieurs stratégies d'élasticité, fournissant une logique pour le contrôle des redimensionnements du système. Précisément, nous avons introduit des stratégies décrivant les différentes méthodes d'élasticité (horizontale, verticale, migration et load balancing). Ces stratégies peuvent s'appliquer en multi-couche, c'est-à-dire, à différents niveaux du système (infrastructure, application, ressources). Elles décrivent comment les actions de reconfigurations (règles de réaction bigraphiques) peuvent être déclenchées afin de permettre au système

Cloud de s'adapter à sa charge de travail de manière réactive, tout en préservant sa cohérence architecturale.

3. **Représentation de l'état élastique des systèmes Cloud.** Nous avons proposé une solution pour la représentation et l'expression des états du point de vue de l'élasticité d'un système Cloud. Nous avons défini un ensemble de prédicats de la logique de premier ordre afin de décrire les différents états d'un système Cloud, du point de vue de son élasticité, tout en couvrant sa configuration en multi-couches. C'est-à-dire en considérant les entités composant le système aux niveau infrastructure et application. Les états identifiés nous ont servis à dégager des états désirables et indésirables en termes d'élasticité. Dans notre approche, ces états élastiques servent à guider les dimensionnements élastiques d'un système Cloud.
4. **Implémentation, exécution autonomique et vérification formelle des stratégies d'élasticité.** Nous avons présenté un encodage des spécifications bigraphiques et des stratégies d'élasticité dans le langage de spécification formelle Maude. Cet encodage permet de préserver la sémantique structurelle des systèmes Cloud tout en l'enrichissant d'aspects quantitatifs et la possibilité d'exprimer les états du système à travers des prédicats de la logique du premier ordre. Maude permet également d'encoder les stratégies d'élasticité afin de permettre leur exécution de manière générique et autonomique. Nous avons procédé à la vérification formelle de l'élasticité des systèmes Cloud modélisés, grâce aux outils fournis par le système Maude. Cette vérification se base sur une technique de model-checking à base d'états, supportée par la logique temporelle linéaire LTL et les structures de *Kripke*.
5. **Évaluation et validation quantitative.** Nous avons proposé une étude expérimentale des stratégies d'élasticité multi-couches introduites. Nous avons validé les comportements élastiques définis d'un point de vue quantitatif, à travers une étude de cas d'un système Cloud existant. Nous avons proposé une approche à base de files d'attente afin de simuler le fonctionnement du système étudié, selon les différentes stratégies d'élasticité introduites. À ces fins, nous avons conçu un outil pour la simulation et le monitoring des comportements élastiques d'un système Cloud, en se basant sur un modèle de files d'attente avec nombre variable de serveurs. Nous avons montré que notre approche d'évaluation pouvait être appliquée en amont du déploiement d'un service basé Cloud. Les résultats de l'étude expérimentale menée ont montré que les stratégies d'élasticité introduites apportent des comportements très riches, pouvant satisfaire plusieurs politiques de haut niveau comme *les hautes performances*, *les hautes économies*, *l'optimisation des coûts d'infrastructure*, *la haute disponibilité des ressources* ou encore *la haute efficience des ressources*. En outre, les fonctionnalités que nous présentons permettent à un fournisseur de Service Cloud de simuler son système afin de prévoir la quantité de ressources à déployer, ainsi que les performances éventuelles de son service. Notre outil peut simuler un système en le confrontant à une infinité de scénarios possibles, basés sur l'intensité et les fluctuations tant prévisibles qu'imprévisibles de la charge de travail.

9.2 Perspectives

Nous identifions ici trois principales perspectives à court, moyen et long termes afin de poursuivre les travaux scientifiques présentés dans ce manuscrit. D'un point de vue pratique, la première perspective se rapport au développement d'un outil complètement

assisté pour concevoir et analyser les systèmes Cloud élastiques. D'un point de vue théorique, la deuxième perspective concerne l'étude de stratégies d'élasticité pour le dimensionnement hybride (horizontal et vertical) de ressources. Enfin, la troisième perspective concerne l'élargissement des stratégies d'élasticité proposées.

Développement d'un outil assisté pour la spécification, la vérification et l'évaluation de l'élasticité. Á court terme, il serait intéressant de développer un outil complètement automatisé et assisté depuis la phase de modélisation jusqu'à l'analyse des résultats obtenus de la vérification des propriétés. Idéalement, l'outil devrait fournir une interface graphique intuitive permettant à un utilisateur de facilement modéliser, selon l'approche bigraphique proposée, ses systèmes Cloud. De cette modélisation, l'idée est de concevoir un traducteur qui permettrait la génération des spécifications Maude à partir des modèles bigraphiques conçus. Enfin, un dernier composant permettrait d'exécuter et d'analyser les comportements élastiques décrits afin de procéder à leur vérification qualitative selon des critères définis par l'utilisateur. Enfin, cet outil permettrait également d'intégrer les capacités de simulation et de monitoring des systèmes Cloud modélisés en vue de faciliter leur administration.

Stratégies d'élasticité pour le dimensionnement Hybride. Á moyen terme, nous visons à fournir une solution pour la spécification et l'analyse de l'élasticité hybride. Cette méthode d'élasticité consiste à combiner les différentes stratégies d'élasticité pour le dimensionnement horizontal et vertical ainsi que les actions de migration et de load balancing. Dans la théorie, le dimensionnement horizontal part du postulat que l'ajout des ressources peut se faire de manière illimitée (i.e. indépendamment des ressources informatiques disponible). Cependant, le dimensionnement vertical est de capacité limitée car il dépend des ressources disponibles. Dans notre approche de modélisation, nous abstrayant la quantité de ressources disponibles (CPU, RAM) via le seuil *max* du serveur, ce qui permet de raisonner sur les capacités de dimensionnement horizontal. D'un autre côté, notre stratégie d'élasticité verticale considère les seuils *max* et *min* qui sont significatifs des ressources disponibles à un haut niveau d'abstraction.

Pour assurer la possibilité d'une élasticité hybride, il faudrait que le dimensionnement vertical impacte le seuil *max* d'une VM et d'un serveur. De manière à mettre à jour leurs capacités en termes d'hébergement de services et VMs en exécution. Pour une VM, l'idée serait de disposer d'une fenêtre glissante de l'intervalle des seuils [*min*–*max*] en réponse aux actions d'adaptation appliquées. D'un autre côté, il serait intéressant de calculer le seuil max d'un serveur en fonction des ressources physiques disponibles à son niveau. Par exemple, il faudrait au minimum une unité de CPU et une unité de RAM pour pouvoir y déployer une VM. De cette manière, ajouter/retirer des ressources (CPU, RAM) à une VM augmenterait/diminuerait alors sa capacité *max* et *min* d'hébergement, tout en diminuant la capacité *max* de son serveur hôte.

Par conséquent, les actions de dimensionnement vertical affecteraient les prédicats déclenchants des stratégies horizontales, puisque les seuils *max* et *min* des entités concernées (VMs et serveurs) seraient modifiés. De la même manière, les actions de dimensionnement horizontal affecteraient les prédicats de déclenchement des stratégies verticale, puisque déployer une nouvelle VM requiert de lui allouer des ressources initiales à partir du serveur hôte. Également, les stratégies de migration et de load balancing accompliraient toujours une tâche complémentaire mais de manière plus optimisée et plus dynamique. Finalement, l'élasticité hybride permettrait d'optimiser au maximum le dimensionnement élastique des

ressources. Si on revient à la courbe décrivant le comportement élastique d'un système (voir Section 2.3). Le dimensionnement élastique hybride permettrait d'éviter au mieux les cas de sur/sous-dimensionnement de l'infrastructure Cloud. Donnant ainsi une courbe qui épouserait presque celle de la demande. Néanmoins, il reste à fournir un important effort de conceptualisation et de formalisation pour concrétiser cette approche. En effet, il n'est pas aisé de fournir une logique correcte afin de gouverner et de coordonner le dimensionnement horizontal et vertical.

Élargissement des stratégies d'élasticité : vers la théorie du contrôle. Á long terme, une de nos principales pistes de recherche consiste à enrichir et raffiner la spécification des stratégies d'élasticité. Dans ce manuscrit, nous avons uniquement présenté des stratégies d'élasticité réactives. Ces stratégies opèrent selon un principe de seuils et les conditions définies sont exprimées dans la logique du premier ordre. Bien que notre approche montre des résultats satisfaisants, il est tout à fait possible d'explorer de toutes nouvelles horizons en termes de possibilités d'adaptation autonomiques, allant jusqu'à étendre et enrichir le concept même de stratégie. Une solution serait de recourir à la théorie du contrôle comme support sémantique afin de concevoir des contrôleurs d'élasticité autonomiques fonctionnant sur le principe de boucles fermées pour la récolte d'information sur le système, et l'application d'actions pour remédier à des symptômes (états) indésirables [Zhu et al., 2009].

La théorie du contrôle permettrait de concevoir des stratégies d'élasticité qui seraient elles-mêmes adaptatives et évolutives en fonction de l'état du système géré. Elle permettrait également de concevoir des contrôleurs hybrides et complexes pour la gestion autonomique de l'élasticité [Mendieta et al., 2017, Li et al., 2009, Dutreilh et al., 2010]. De tels contrôleurs permettraient non seulement de fournir des comportements hybrides en termes de dimensionnement horizontal et vertical, mais également la définition de stratégies réactives et proactives. Le but final d'une approche à base de la théorie du contrôle serait de fournir des comportements élastiques permettant une adaptation d'une souplesse optimale, amenant idéalement à la suppression définitive des états de sur et sous dimensionnement d'un système Cloud élastique. Une telle solution permettrait d'approvisionner les ressources de manière exactement proportionnelle à la demande reçue, tout en étant à même de s'adapter efficacement sur le court terme, mais également de prédire et se préparer à gérer des évènements futurs, tels qu'une augmentation ou une baisse importante de la charge de travail en entrée.

Bibliographie

[Aceto et al., 2013] Aceto, G., Botta, A., De Donato, W., and Pescapè, A. (2013). Cloud monitoring : A survey. *Computer Networks*, 57(9) :2093–2115.

[Al-Dhuraibi, 2018] Al-Dhuraibi, Y. (2018). *Flexible Framework for Elasticity in Cloud Computing*. PhD Thesis, Université lille1.

[Ali-Eldin et al., 2012] Ali-Eldin, A., Tordsson, J., and Elmroth, E. (2012). An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212. IEEE.

[Amazon, 2019] Amazon (2019). Tarification des instances EC2 – Amazon Web Services (AWS). `https://aws.amazon.com/fr/ec2/pricing/on-demand/`. [Consulté en ligne le 2019-01-11].

[Amziani, 2015] Amziani, M. (2015). *Modeling, evaluation and provisioning of elastic service-based business processes in the cloud*. Theses, Institut National des Télécommunications.

[Andrikopoulos et al., 2013] Andrikopoulos, V., Binz, T., Leymann, F., and Strauch, S. (2013). How to adapt applications for the cloud environment. *Computing*, 95(6) :493–535.

[Baier and Katoen, 2008] Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

[Baynat, 2000] Baynat, B. (2000). Théorie des files d'attente. *Hermès, Paris.*

[Benzadri et al., 2013] Benzadri, Z., Belala, F., and Bouanaka, C. (2013). Towards a formal model for cloud computing. In *International Conference on Service-Oriented Computing*, pages 381–393. Springer.

[Berns and Ghosh, 2009] Berns, A. and Ghosh, S. (2009). Dissecting self-* properties. In *Self-Adaptive and Self-Organizing Systems, 2009. SASO'09. Third IEEE International Conference on*, pages 10–19. IEEE.

[Berry, 2006] Berry, R. (2006). QUEUING THEORY. *Senior Project Archive*, page 1.

[Bersani et al., 2014] Bersani, M. M., Bianculli, D., Dustdar, S., Gambi, A., Ghezzi, C., and Krstić, S. (2014). Towards the formalization of properties of cloud-based elastic systems. In *Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems*, pages 38–47. ACM.

[Birkedal et al., 2007] Birkedal, L., Damgaard, T. C., Glenstrup, A. J., and Milner, R. (2007). Matching of bigraphs. *Electronic Notes in Theoretical Computer Science*, 175(4) :3–19.

[Borenstein and Blake, 2011] Borenstein, N. and Blake, J. (2011). Cloud computing standards : Where's the beef ? *IEEE Internet Computing*, 15(3) :74–78.

[Bósa et al., 2015] Bósa, K., Holom, R.-M., and Vleju, M. B. (2015). A formal model of client-cloud interaction. In *Correct Software in Web Applications and Web Services*, pages 83–144. Springer.

[Bundgaard et al., 2008] Bundgaard, M., Glenstrup, A. J., Hildebrandt, T., Højsgaard, E., and Niss, H. (2008). Formalizing WS-BPEL and higher order mobile embedded business processes in the bigraphical programming languages (BPL) tool. *Copenhagen*.

[Buyya et al., 2008] Buyya, R., Yeo, C. S., and Venugopal, S. (2008). Market-oriented cloud computing : Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13. Ieee.

[Calliope, 2016] Calliope (2016). Cloud : les modèles de déploiement. https ://blog.3li.com/cloud-les-modeles-de-deploiement/. [Consulté en ligne le 2019-01-11].

[Carlson et al., 2012] Carlson, A. B., Imwalle, G. P., and Kowalski, T. R. (2012). *Data center with low power usage effectiveness*. Google Patents.

[Carrasco et al., 2017] Carrasco, J., Durán, F., and Pimentel, E. (2017). Runtime Migration of Applications in a Trans-Cloud Environment. In *International Conference on Service-Oriented Computing*, pages 55–66. Springer.

[Chan, 2014] Chan, W. (2014). Optimisation des horaires des agents et du routage des appels dans les centres d'appels.

[Chechik and Gannon, 2001] Chechik, M. and Gannon, J. (2001). Automatic analysis of consistency between requirements and designs. *IEEE transactions on Software Engineering*, 27(7) :651–672.

[Clavel et al., 2016] Clavel, M., Durán, F., Eker, S., Escobar, S., Lincoln, P., Martı-Oliet, N., Meseguer, J., and Talcott, C. (2016). Maude Manual (Version 2.7. 1).

[Clavel et al., 2002] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martı-Oliet, N., Meseguer, J., and Quesada, J. F. (2002). Maude : Specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2) :187–243.

[Clavel et al., 2007] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. (2007). *All about maude-a high-performance logical framework : how to specify, program and verify systems in rewriting logic*. Springer-Verlag.

[Clavel et al., 1996] Clavel, M., Eker, S., Lincoln, P., and Meseguer, J. (1996). Principles of maude. *Electronic Notes in Theoretical Computer Science*, 4 :65–89.

[Cloud, 2011] Cloud, A. E. C. (2011). Amazon web services. *Retrieved November*, 9 :2011.

[Copeland et al., 2015] Copeland, M., Soh, J., Puca, A., Manning, M., and Gollob, D. (2015). Microsoft azure and cloud computing. In *Microsoft Azure*, pages 3–26. Springer.

[Copil et al., 2013] Copil, G., Moldovan, D., Truong, H.-L., and Dustdar, S. (2013). Multi-level elasticity control of cloud services. In *International Conference on Service-Oriented Computing*, pages 429–436. Springer.

[Dashti and Rahmani, 2016] Dashti, S. E. and Rahmani, A. M. (2016). Dynamic VMs placement for energy efficiency by PSO in cloud computing. *Journal of Experimental & Theoretical Artificial Intelligence*, 28(1-2) :97–112.

[Docker, 2019] Docker (2019). Docker. https ://www.docker.com/index.html. [Consulté en ligne le 2019-02-22].

[Dupont, 2016] Dupont, S. (2016). *Gestion autonomique de l'élasticité multi-couche des applications dans le Cloud : vers une utilisation efficiente des ressources et des services du Cloud*. PhD thesis, Ecole des Mines de Nantes.

[Dutreilh et al., 2010] Dutreilh, X., Moreau, A., Malenfant, J., Rivierre, N., and Truck, I. (2010). From data center resource allocation to control theory and back. In *2010 IEEE 3rd international conference on cloud computing*, pages 410–417. IEEE.

[Ellson et al., 2001] Ellson, J., Gansner, E., Koutsofios, L., North, S. C., and Woodhull, G. (2001). Graphviz—open source graph drawing tools. In *International Symposium on Graph Drawing*, pages 483–484. Springer.

[Elsborg, 2009] Elsborg, E. (2009). *Bigraphs : Modelling, Simulation, and Type Systems*. PhD Thesis, IT-Universitetet i København.

[Erlang, 2019] Erlang (2019). Erlang C Formula Online Calculator. http ://www-ens.iro.umontreal.ca/ chanwyea/erlang/erlangC.html. [Consulté en ligne le 2019-02-20].

[Faithfull et al., 2013] Faithfull, A. J., Perrone, G., and Hildebrandt, T. T. (2013). Big red : A development environment for bigraphs. *Electronic Communications of the EASST*, 61.

[Fehling et al., 2014] Fehling, C., Leymann, F., Retter, R., Schupeck, W., and Arbitter, P. (2014). *Cloud Computing Patterns : Fundamentals to Design, Build, and Manage Cloud Applications*. Springer.

[Firdhous et al., 2011] Firdhous, M., Ghazali, O., and Hassan, S. (2011). Modeling of cloud system using Erlang formulas. In *Communications (APCC), 2011 17th Asia-Pacific Conference on*, pages 411–416. IEEE.

[Fox et al., 2009] Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and Stoica, I. (2009). Above the clouds : A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13) :2009.

[Freitas et al., 2012] Freitas, A. L., Parlavantzas, N., and Pazat, J.-L. (2012). An integrated approach for specifying and enforcing slas for cloud services. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 376–383. IEEE.

[Galante and Bona, 2012] Galante, G. and Bona, L. C. E. d. (2012). A survey on cloud computing elasticity. In *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, pages 263–270. IEEE Computer Society.

[Gambi et al., 2016] Gambi, A., Pezze, M., and Toffetti, G. (2016). Kriging-based self-adaptive cloud controllers. *IEEE Transactions on Services Computing*, 9(3) :368–381.

[Garfinkel, 1999] Garfinkel, S. (1999). *Architects of the information society : 35 years of the Laboratory for Computer Science at MIT*. MIT press.

[Google, 2019] Google (2019). Gmail : It's cooler in the cloud. https ://cloud.googleblog.com/2011/09/gmail-its-cooler-in-cloud.html. [Consulté en ligne le 2019-02-22].

[Group and others, 2005] Group, I. B. M. and others (2005). An architectural blueprint for autonomic computing. *IBM White paper.*

[Gueroui, 2015] Gueroui, A. (2015). Les files d'attente. Cours, Université de Versailles St-Quentin-en-Yvelines.

[Herbst et al., 2013] Herbst, N. R., Kounev, S., and Reussner, R. H. (2013). Elasticity in Cloud Computing : What It Is, and What It Is Not. In *ICAC*, volume 13, pages 23–27.

[Højsgaard and Glenstrup, 2011] Højsgaard, E. and Glenstrup, A. J. (2011). The bpl tool : A tool for experimenting with bigraphical reactive systems. *Bigraphical Languages and their Simulation*, page 85.

[Horn, 2001] Horn, P. (2001). Autonomic computing : IBM$\backslash$'s Perspective on the State of Information Technology.

[Huebscher and McCann, 2008] Huebscher, M. C. and McCann, J. A. (2008). A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys (CSUR)*, 40(3) :7.

[Jacob et al., 2004] Jacob, B., Lanyon-Hogg, R., Nadgir, D. K., and Yassin, A. F. (2004). A practical guide to the ibm autonomic computing toolkit. *IBM Redbooks*, 4(10).

[Jamshidi et al., 2014] Jamshidi, P., Ahmad, A., and Pahl, C. (2014). Autonomic resource provisioning for cloud-based software. In *Proceedings of the 9th international symposium on software engineering for adaptive and self-managing systems*, pages 95–104. ACM.

[Joustra and Van Dijk, 2001] Joustra, P. E. and Van Dijk, N. M. (2001). Simulation of check-in at airports. In *Proceedings of the 33nd conference on Winter simulation*, pages 1023–1028. IEEE Computer Society.

[Kaur and Chana, 2014] Kaur, P. D. and Chana, I. (2014). Cloud based intelligent system for delivering health care as a service. *Computer methods and programs in biomedicine*, 113(1) :346–359.

[Kendall, 1953] Kendall, D. G. (1953). Stochastic processes occurring in the theory of queues and their analysis by the method of the imbedded Markov chain. *The Annals of Mathematical Statistics*, pages 338–354.

[Kephart and Chess, 2003] Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, (1) :41–50.

[Khebbeb et al., 2018a] Khebbeb, K., Hameurlain, N., and Belala, F. (2018a). Modeling and evaluating cross-layer elasticity strategies in cloud systems. In *International Conference on Model and Data Engineering*, pages 168–183. Springer.

[Khebbeb et al., 2018b] Khebbeb, K., Hameurlain, N., Belala, F., and Sahli, H. (2018b). Formal modelling and verifying elasticity strategies in cloud systems. *IET Software, The Institution of Engineering and Technology*, 13(1) :25–35.

[Khebbeb et al., 2017] Khebbeb, K., Sahli, H., Hameurlain, N., and Belala, F. (2017). A brs based approach for modeling elastic cloud systems. In *International Conference on Service-Oriented Computing*, pages 5–17. Springer.

[Kikuchi and Hiraishi, 2014] Kikuchi, S. and Hiraishi, K. (2014). Improving reliability in management of cloud computing infrastructure by formal methods. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–7. IEEE.

[Kleinrock, 1976] Kleinrock, L. (1976). *Queueing systems, volume 2 : Computer applications*, volume 66. wiley New York.

[Kleinrock, 2005] Kleinrock, L. (2005). A vision for the Internet. *ST Journal of Research*, 2(1) :4–5.

[Koomey, 2011] Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, 9.

[Kouki and Ledoux, 2013] Kouki, Y. and Ledoux, T. (2013). Rightcapacity : Sla-driven cross-layer cloud elasticity management. *International Journal of Next-Generation Computing*, 4(3).

[Kranas et al., 2012] Kranas, P., Anagnostopoulos, V., Menychtas, A., and Varvarigou, T. (2012). Elaas : An innovative elasticity as a service framework for dynamic management across the cloud stack layers. In *2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, pages 1042–1049. IEEE.

[Krivine et al., 2008] Krivine, J., Milner, R., and Troina, A. (2008). Stochastic bigraphs. *Electronic Notes in Theoretical Computer Science*, 218 :73–96.

[Letondeur, 2014] Letondeur, L. (2014). *Planification pour la gestion autonomique de l'élasticité d'applications dans le cloud*. PhD Thesis, Université de Grenoble.

[Li et al., 2009] Li, Q., Hao, Q., Xiao, L., and Li, Z. (2009). Adaptive management of virtualized resources in cloud computing using feedback control. In *2009 First International Conference on Information Science and Engineering*, pages 99–102. IEEE.

[Loff and Garcia, 2014] Loff, J. and Garcia, J. (2014). Vadara : Predictive elasticity for cloud applications. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 541–546. IEEE.

[Lorido-Botrán et al., 2012] Lorido-Botrán, T., Miguel-Alonso, J., and Lozano, J. A. (2012). Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, 12 :2012.

[Mastelic et al., 2015] Mastelic, T., Oleksiak, A., Claussen, H., Brandic, I., Pierson, J.-M., and Vasilakos, A. V. (2015). Cloud computing : Survey on energy efficiency. *Acm computing surveys (csur)*, 47(2) :33.

[Mazalov and Gurtov, 2012] Mazalov, V. V. and Gurtov, A. (2012). Queuing system with on-demand number of servers. *Mathematica Applicanda*, 40(2) :1–12.

[Mell et al., 2011] Mell, P., Grance, T., and others (2011). The NIST definition of cloud computing.

[Mendieta et al., 2017] Mendieta, M., Martín, C. A., and Abad, C. L. (2017). A control theory approach for managing cloud computing resources : A proof-of-concept on memory partitioning. In *Ecuador Technical Chapters Meeting (ETCM), 2017 IEEE*, pages 1–6. IEEE.

[Microsoft, 2019] Microsoft (2019). Pricing - Windows Virtual Machines | Microsoft Azure. `https://azure.microsoft.com/en-us/pricing/details/virtual-machines/windows/`. [Consulté en ligne le 11/01/2019].

[Milner, 1980] Milner, R. (1980). A calculus of communicating systems. *LNCS*, 92.

[Milner, 1993] Milner, R. (1993). Action calculi, or syntactic action structures. In *International Symposium on Mathematical Foundations of Computer Science*, pages 105–121. Springer.

[Milner, 2001a] Milner, R. (2001a). Bigraphical reactive systems. In *International Conference on Concurrency Theory*, pages 16–35. Springer.

[Milner, 2001b] Milner, R. (2001b). Bigraphical reactive systems : basic theory. Technical report, University of Cambridge, Computer Laboratory.

[Milner, 2005] Milner, R. (2005). Axioms for bigraphical structure. *Mathematical Structures in Computer Science*, 15(6) :1005–1032.

[Milner, 2006] Milner, R. (2006). Pure bigraphs : Structure and dynamics. *Information and computation*, 204(1) :60–122.

[Milner, 2008] Milner, R. (2008). Bigraphs and their algebra. *Electronic Notes in Theoretical Computer Science*, 209(0) :5–19.

[Milner, 2009] Milner, R. (2009). *The space and motion of communicating agents*. Cambridge University Press.

[Milner et al., 1992] Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, i. *Information and computation*, 100(1) :1–40.

[Milner et al., 1997] Milner, R., Tofte, M., Harper, R., and MacQueen, D. (1997). *The definition of standard ML : revised*. MIT press.

[Młyńczak, 2007] Młyńczak, M. (2007). Queuing system with variable server number. *W SKRÓCIE*, page 63.

[Moldovan et al., 2015] Moldovan, D., Copil, G., Truong, H.-L., and Dustdar, S. (2015). MELA : elasticity analytics for cloud services. *International Journal of Big Data Intelligence*, 2(1) :45–62.

[Montgomery et al., 1990] Montgomery, D. C., Johnson, L. A., and Gardiner, J. S. (1990). *Forecasting and time series analysis*. McGraw-Hill New York etc.

[Nosek Jr and Wilson, 2001] Nosek Jr, R. A. and Wilson, J. P. (2001). Queuing theory and customer satisfaction : a Review of terminology, trends, and applications to pharmacy practice. *Hospital pharmacy*, 36(3) :275–279.

[OpenShift, 2019] OpenShift (2019). OpenShift : Container Application Platform by Red Hat, Built on Docker and Kubernetes. https ://www.openshift.com. [Consulté en ligne le 2019-02-22].

[Patel et al., 2009] Patel, P., Ranabahu, A. H., and Sheth, A. P. (2009). Service level agreement in cloud computing.

[Perrone et al., 2012] Perrone, G., Debois, S., and Hildebrandt, T. T. (2012). A model checker for bigraphs. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 1320–1325. ACM.

[Phillips, 2004] Phillips, J. (2004). Simulation and Queuing Theory. Technical report, Heriot-Watt University, Edinburgh, Ecosse.

[Poniatowski, 2009] Poniatowski, M. (2009). *Foundation of Green IT : Consolidation, virtualization, efficiency, and ROI in the data center*. Pearson Education.

[Rackspace, 2010] Rackspace, U. (2010). *Inc.,"The Rackspace Cloud,"*.

[Rady, 2013] Rady, M. (2013). Formal definition of service availability in cloud computing using OWL. In *International Conference on Computer Aided Systems Theory*, pages 189–194. Springer.

[Sabharwal et al., 2013] Sabharwal, M., Agrawal, A., and Metri, G. (2013). Enabling green it through energy-aware software. *IT Professional*, 15(1) :19–27.

[Sahli, 2017] Sahli, H. (2017). *A Formal Framework for Modelling and Verifying Cloud-based Elastic Systems*. Theses, Université Constantine 2 - Abdelhamid Mehri.

[Scalas and Bartoletti, 2015] Scalas, A. and Bartoletti, M. (2015). The LTS WorkBench. *arXiv preprint arXiv :1508.04853*.

[Schaffer, 2009] Schaffer, H. E. (2009). X as a service, cloud computing, and the need for good judgment. *IT professional*, 11(5).

[Schoren, 2011] Schoren, R. (2011). Correspondence between kripke structures and labeled transition systems for model minimization. In *Seminar project, Technische Universiteit Eindhoven, Department of Computer Science*.

[Sefraoui et al., 2012] Sefraoui, O., Aissaoui, M., and Eleuldj, M. (2012). OpenStack : toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3) :38–42.

[Sevegnani and Calder, 2015] Sevegnani, M. and Calder, M. (2015). Bigraphs with sharing. *Theoretical Computer Science*, 577 :43–73.

[Sevegnani and Calder, 2016] Sevegnani, M. and Calder, M. (2016). BigraphER : rewriting and analysis engine for bigraphs. In *International Conference on Computer Aided Verification*, pages 494–501. Springer.

[Smith and Nair, 2005] Smith, J. E. and Nair, R. (2005). The architecture of virtual machines. *Computer*, 38(5) :32–38.

[Souri et al., 2018] Souri, A., Navimipour, N. J., and Rahmani, A. M. (2018). Formal verification approaches and standards in the cloud computing : a comprehensive and systematic review. *Computer Standards & Interfaces*, 58 :1–22.

[Steam, 2019] Steam (2019). Steam, la plateforme ultime de jeu en ligne. https ://store.steampowered.com/about/. [Consulté en ligne le 2019-02-03].

[SteamSpy, 2019] SteamSpy (2019). Steamspy - all the data about steam games. https ://steamspy.com/year/. [Consulté en ligne le 2019-02-05].

[Suleiman et al., 2012] Suleiman, B., Sakr, S., Jeffery, R., and Liu, A. (2012). On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure. *Journal of Internet Services and Applications*, 3(2) :173–193.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

[Trihinas et al., 2014] Trihinas, D., Sofokleous, C., Loulloudes, N., Foudoulis, A., Pallis, G., and Dikaiakos, M. D. (2014). Managing and monitoring elastic cloud applications. In *International Conference on Web Engineering*, pages 523–527. Springer.

[Uriarte et al., 2014] Uriarte, R. B., Tiezzi, F., and Nicola, R. D. (2014). Slac : A formal service-level-agreement language for cloud computing. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 419–426. IEEE Computer Society.

[Verma et al., 2008a] Verma, A., Ahuja, P., and Neogi, A. (2008a). pMapper : power and migration cost aware application placement in virtualized systems. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 243–264. Springer-Verlag New York, Inc.

[Verma et al., 2008b] Verma, A., Ahuja, P., and Neogi, A. (2008b). Power-aware dynamic placement of hpc applications. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 175–184. ACM.

[Voorsluys et al., 2009] Voorsluys, W., Broberg, J., Venugopal, S., and Buyya, R. (2009). Cost of virtual machine live migration in clouds : A performance evaluation. In *IEEE International Conference on Cloud Computing*, pages 254–265. Springer.

[Wang et al., 2010] Wang, L., Von Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J., and Fu, C. (2010). Cloud computing : a perspective study. *New Generation Computing*, 28(2) :137–146.

[Wang et al., 2012] Wang, W.-J., Lo, Y.-M., Chen, S.-J., and Chang, Y.-S. (2012). Intelligent application migration within a self-provisioned hybrid cloud environment. In *Computer Science and Convergence*, pages 295–303. Springer.

[Weber et al., 2014] Weber, A., Herbst, N., Groenda, H., and Kounev, S. (2014). Towards a resource elasticity benchmark for cloud environments. In *Proceedings of the 2nd International Workshop on Hot Topics in Cloud service Scalability*, page 5. ACM.

[Weiss et al., 2011] Weiss, G., Zeller, M., and Eilers, D. (2011). Towards automotive embedded systems with self-x properties. In *New Trends and Developments in Automotive System Engineering*. InTech.

[Yataghene et al., 2014] Yataghene, L., Amziani, M., Ioualalen, M., and Tata, S. (2014). A queuing model for business processes elasticity evaluation. In *Advanced Information Systems for Enterprises (IWAISE), 2014 International Workshop on*, pages 22–28. IEEE.

[Yazdanov and Fetzer, 2012] Yazdanov, L. and Fetzer, C. (2012). Vertical scaling for prioritized vms provisioning. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 118–125. IEEE.

[Zhang et al., 2010] Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing : state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1) :7–18.

[Zhu et al., 2009] Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Padala, P., and Shin, K. (2009). What does control theory bring to systems research ? *SIGOPS Oper. Syst. Rev.*, 43(1) :62–69.