Optimization of autonomic resources for the management of service-based business processes in the Cloud

Leila Hadded

université PARIS-SACLAY

TELECOM SudParis

FACULTE DES SCIENCES DE TUNIS

NNT : 2018SACLL006

Thèse de doctorat

Optimisation des ressources autonomiques pour la gestion des processus métier à base de services dans le Cloud

Thèse de doctorat de l'Université Paris-Saclay
préparée à Télécom SudParis et la Faculté des Sciences de Tunis

Ecole doctorale n°580: Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Tunis, le 06-10-2018, par

Leila Hadded

Composition du Jury :

Habib Ounelli Professeur, Faculté des Sciences de Tunis, Tunisie	Président
Xavier Blanc Professeur, Université de Bordeaux, France	Rapporteur
Leila Jemni Ben Ayed Professeur, Ecole Nationale des Sciences de l'Informatique, Tunisie	Rapporteur
Hanna Klaudel Professeur, Université d'Evry, France	Examinatrice
Samir Tata Professeur, Télécom SudParis (Samovar), France	Directeur de thèse
Faouzi Ben Charrada Professeur, Faculté des Sciences de Tunis (LIMTIC), Tunisie	Co-directeur de thèse

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Optimisation des ressources autonomiques pour la gestion des processus métier à base de services dans le Cloud

Mots clés: Cloud Computing, Informatique autonome, Processus métier à base de services, Allocation de resources Cloud, Programmation linéaire

Résumé: Le Cloud Computing est un nouveau paradigme qui fournit des ressources informatiques sous forme de services à la demande via internet. Il est fondé sur le modèle de facturation pay-per-use et il est de plus en plus utilisé pour le déploiement et l'exécution des processus métier en général et des processus métier à base de services (SBPs) en particulier. Les environnements Cloud sont généralement très dynamiques. À cet effet, il devient indispensable de s'appuyer sur des agents intelligents appelés gestionnaires autonomiques (AMs), qui permettent de rendre les SBPs capables de se gérer de façon autonome afin de faire face aux changements dynamiques induits par le Cloud. Cependant, les solutions existantes sont limitées à l'utilisation soit d'un AM centralisé, soit d'un AM par service pour la gestion d'un SBP. Il est évident que la deuxième solution représente un gaspillage d'AMs et peut conduire à la prise de décisions de gestion contradictoires, tandis que la première solution peut conduire à des goulots d'étranglement au niveau de la gestion du SBP. Par conséquent, il est essentiel de trouver le nombre optimal d'AMs qui seront utilisés pour gérer le SBP afin de minimiser leur nombre tout en évitant les goulots d'étranglement. De plus, en raison de l'hétérogénéité des ressources Cloud et de la diversité de la qualité de service (QoS) requise par les SBPs, l'allocation des ressources Cloud pour ces AMs peut entraîner des coûts de calcul et de communication élevés et/ou une QoS inférieure à celle exigée. Pour cela, il est également essentiel de trouver l'allocation optimale des ressources Cloud pour les AMs qui seront utilisés pour gérer un SBP afin de minimiser les coûts tout en maintenant les exigences de QoS. Dans ce travail, nous proposons un modèle d'optimisation déterministe pour chacun de ces deux problèmes. En outre, en raison du temps nécessaire pour résoudre chacun de ces problèmes qui croît de manière exponentielle avec la taille du problème, nous proposons des algorithmes quasi-optimaux qui permettent d'obtenir de bonnes solutions dans un temps raisonnable.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Title: Optimization of Autonomic Resources for the Management of Service-based Business Processes in the Cloud

Keywords: Cloud Computing, Autonomic Computing, Service-based business process, Cloud resource allocation, Linear programming

Abstract: Cloud Computing is a new paradigm that provides computing resources as a service over the Internet in a pay-per-use model. It is increasingly used for hosting and executing business processes in general and Service-based Business Processes (SBPs) in particular. Cloud environments are usually highly dynamic. Hence, executing these SBPs requires autonomic management to cope with the changes in Cloud environments, which implies the usage of a number of controlling devices, referred to as Autonomic Managers (AMs). However, existing solutions are limited to using either a centralized AM or an AM per service for managing a whole SBP. It is obvious that the latter solution is resource-consuming and may lead to conflicting management decisions, while the former may lead to management bottlenecks. A main problem in this context consists in finding the optimal number of AMs for the management of an SBP, minimizing costs in terms of number of AMs while at the same time avoiding management bottlenecks and ensuring good management performance. Moreover, due to the heterogeneity of Cloud resources and the diversity of the required Quality of Service (QoS) of SBPs, the allocation of Cloud resources to these AMs may result in high computing costs and an increase in the communication overheads and/or lower QoS. It is also crucial to find the optimal allocation of Cloud resources to the AMs, minimizing costs while maintaining the QoS requirements. To address these challenges, we propose in this work, a deterministic optimization model for each problem. Furthermore, due to the amount of time needed to solve each one of these problems, which grows exponentially with the size of the problem, we propose near-optimal algorithms that provide good solutions in a reasonable time.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

To the memory of my aunt Khadija,
To my parents and lovely sister

Acknowledgements

I would like to thank the members of the thesis committee. I thank the committee president Professor Habib Ounelli for accepting to chair my thesis defense, the reading committee members Professor Xavier Blanc and Professor Leila Jemni Ben Ayed for their interest and Professor Hanna Klaudel for evaluating my work and being part of the defense committee.

I would like to express my deepest gratitude to my advisors, Professor Faouzi Ben Charrada and Professor Samir Tata, for their generous support without which this thesis would not have been possible. Despite their many responsibilities, they have always found time to provide feedbacks, new ideas and suggestions on my work. I thank them for allowing me to grow as a research scientist.

I owe my deepest appreciation to the members of the Computer Science department at Telecom SudParis. I would like to thank Brigitte Houassine for her kind help and assistance. Many thanks to Aicha, Nabila, Amina, Hayet, and Rania for the lovely moments we spent.

A special thank to my friends Sarra, Maweheb, and Anouer for their emotional support and for all their help.

I am forever thankful to my family: my father, my mother, and my sister who were always there for me with encouraging words whenever I started doubting myself. Your encouragement made me go forward and made me want to succeed. Thank you so much for having faith in me. I love you so much.

List of Publications[1]

[1] Leila Hadded, Faouzi Ben Charrada, and Samir Tata, *Optimizing Autonomic Resources for the Management of Large Service-Based Business Processes*, IEEE Transactions on Services Computing (IEEE TSC), Accepted for publication, 2018 **(Impact Factor: 4.418)**.

[2] Leila Hadded, Faouzi Ben Charrada, and Samir Tata, *Efficient Resource Allocation for Autonomic Service-Based Applications in the Cloud*, 15th IEEE International Conference on Autonomic Computing (IEEE ICAC), Trento, Italy, September 3-7, 2018 **(Ranking: B)**.

[3] Leila Hadded, Faouzi Ben Charrada, and Samir Tata, *Optimization of Autonomic Manager Components in Service-Based Applications*, 14th IEEE International Conference on Services Computing (IEEE SCC), Honolulu, HI, USA, June 25-30, 2017 **(Ranking: A)**.

[4] Leila Hadded, Faouzi Ben Charrada, and Samir Tata, *Optimization and Approximate Placement of Autonomic Resources for the Management of Service-Based Applications in the Cloud*, 24th International Conference on Cooperation Information Systems(CoopIS), Rhodes, Greece, October 24-28, 2016 **(Ranking: A)**.

[5] Leila Hadded, Faouzi Ben Charrada, and Samir Tata, *An Efficient Optimization Algorithm of Autonomic Managers in Service-Based Applications*, 23th International Conference on Cooperation Information Systems(CoopIS), Rhodes, Greece, October 26-30, 2015 **(Ranking: A)**.

[1] The conferences ranking is based on the CORE 2018 classification available at http://portal.core.edu.au/conf-ranks/

Contents

1 Introduction **1**
- 1.1 General Context . . . 1
- 1.2 Motivation and Problem Statement . . . 2
- 1.3 Objectives and Contributions . . . 3
  - 1.3.1 AMs Optimization in SBPs . . . 3
  - 1.3.2 AMs Components Optimization in Timed SBPs . . . 4
  - 1.3.3 Placement of AMs for the Management of SBPs in the Cloud . . . 4
- 1.4 Thesis Outline . . . 4

2 Background & State of the Art **6**
- 2.1 Introduction . . . 6
- 2.2 Background . . . 7
  - 2.2.1 Cloud Computing . . . 7
  - 2.2.2 Service-Oriented Architecture and Service-based Business Processes 9
  - 2.2.3 Autonomic Computing . . . 10
  - 2.2.4 Optimization Problems . . . 11
- 2.3 State of the Art . . . 13
  - 2.3.1 Evaluation Criteria . . . 13
  - 2.3.2 Autonomic Computing . . . 13
  - 2.3.3 Optimal Resource Allocation . . . 18
- 2.4 Conclusion . . . 23

3 Autonomic Managers Optimization in SBPs **24**
- 3.1 Introduction . . . 24
- 3.2 Problem Description and Formulation . . . 25
  - 3.2.1 SBP Modeling . . . 25
  - 3.2.2 Problem Statement . . . 26
  - 3.2.3 Problem Formulation . . . 27
  - 3.2.4 Need for Approximate Approach . . . 29

3.3 AMs Optimization in SBP Graphs with Cycles . . . 29
3.3.1 Proposed Approach . . . 29
3.3.2 Illustrative Example . . . 33
3.3.3 Performance Evaluation . . . 34
3.4 AMs Optimization in SBP Graphs Without Cycles . . . 38
3.4.1 Proposed Approach . . . 38
3.4.2 Illustrative Example . . . 41
3.4.3 Performance Evaluation . . . 44
3.5 Conclusion . . . 50

4 Autonomic Managers Components Optimization in Timed SBPs 51
4.1 Introduction . . . 51
4.2 Preliminaries: Monitor, Analyzer, Planner, Executor . . . 52
4.3 Problem Description and Formulation . . . 53
4.3.1 Problem Description . . . 53
4.3.2 Timed SBP Modeling . . . 54
4.3.3 Notations and Assumptions . . . 55
4.3.4 Problem Formulation . . . 56
4.4 Proposed Approach . . . 58
4.4.1 Approach Overview . . . 58
4.4.2 ParallelServices Algorithm . . . 58
4.4.3 M-A-P-E_Assignment Algorithm . . . 60
4.4.4 Theoretical Complexity . . . 66
4.5 Illustrative Example . . . 67
4.6 Performance Evaluation . . . 71
4.7 Conclusion . . . 75

5 Placement of Autonomic Managers for the management of SBPs in the Cloud 76
5.1 Introduction . . . 77
5.2 Preliminaries: IaaS Cloud provider, Deployed SBP . . . 77
5.2.1 IaaS Cloud Provider . . . 77
5.2.2 Deployed SBP . . . 78
5.3 Problem Description . . . 79
5.4 Placement of AMs in the Cloud for Efficient Management of SBPs . . . 79
5.4.1 Problem Formulation . . . 80
5.4.2 Proposed Algorithm . . . 81
5.4.3 Illustrative Example . . . 82
5.4.4 Experimental Results . . . 84
5.5 Joint Optimization of the Number of AMs and their Placement in the Cloud for Efficient Management of SBPs . . . 86
5.5.1 Problem Formulation . . . 86

5.5.2 Proposed Approach 91
5.5.3 Illustrative Example 95
5.5.4 Experimental Results 96
5.6 Conclusion 98

6 Conclusion and Future Works **99**
6.1 Contributions 99
6.2 Future work 100

List of Figures

2.1 Cloud computing models [1]. 8
2.2 SOA actors and roles. 9
2.3 IBM architecture of autonomic resource [2]. 11
2.4 Structure of autonomic resources and their interactions [3]. 14
2.5 System architecture for autonomic Cloud management [4]. 15
2.6 Rainbow framework [5]. 15
2.7 StarMX high-level static architecture [6]. 16
2.8 Autonomic component "A" [7]. 17
2.9 Architecture overview of self-adaptation framework [8]. 18

3.1 First research problem: Finding appropriate number of AMs for management of SBP services. 27
3.2 Example of SBP graph with cycle. 33
3.3 First proposal: Number of AMs versus number of services - Experiments on IBM and SAP datasets. 36
3.4 First proposal: Lower bound number of AMs / services number (in %) versus SBP graph density - Experiments on randomly generated dataset. . 37
3.5 First proposal: Cardinality of ParallelSets versus number of services - Experiments on randomly generated dataset. 37
3.6 First proposal: Difference between number of AMs provided by AMsAssignement algorithm and lower bound number of AMs (in %) - Experiments on randomly generated dataset. 38
3.7 Example of SBP graph without cycles where XOR-Split preceding two AND-Splits (left) and result of applying Algorithms 3.4 and 3.5 on it. . . 43
3.8 Example of SBP graph without cycles where AND-Split preceding two XOR-Splits (left) and result of applying Algorithms 3.4 and 3.5 on it. . . 44
3.9 Second proposal: Number of AMs versus number of services - Experiments on IBM and SAP datasets. 46

3.10 Second proposal: Execution time versus number of services - Experiments on IBM and SAP datasets. . . . 47
3.11 Second proposal: Number of AMs versus number of services - Experiments on randomly generated dataset. . . . 48
3.12 Second proposal: Execution time versus number of services - Experiments on randomly generated dataset. . . . 49

4.1 Autonomic control loop for Cloud resource. . . . 52
4.2 Example of timed SBP graph. . . . 55
4.3 Example of timed SBP graph. . . . 67
4.4 Execution trace of ParallelServices algorithm. . . . 68
4.5 Execution trace of P-E_Assignment algorithm. . . . 69
4.6 Execution trace of M/A_Assignment algorithm: Assignement of analyzers to SBP services. . . . 70
4.7 Execution trace of M/A_Assignment algorithm: Assignement of monitors to SBP services. . . . 71
4.8 Number of monitors, analyzers, planners, and executors versus number of services varying from 10 to 26. . . . 73
4.9 Number of monitors, analyzers, planners, and executors versus number of services varying from 30 to 100. . . . 74
4.10 Execution time versus number of services. . . . 74

5.1 Cloud infrastructure provider [9]. . . . 77
5.2 Second research problem: Finding the best placement decisions of AMs to be used by SBP services in the Cloud. . . . 79
5.3 Example of SBP composed of six services deployed on three different homogeneous VMs. . . . 82
5.4 First proposal: AMs placement cost versus number of services and VMs. . 85
5.5 First proposal: Execution time of CPLEX versus number of services. . . . 85
5.6 Example of SBP composed of five services deployed on three different heterogeneous VMs. . . . 95
5.7 Second proposal: Placement cost of AM components and their number versus number of services. . . . 97
5.8 Second proposal: Execution time of CPLEX versus number of services. . . 98
5.9 Second proposal: Execution time of proposed approach versus VMs capacities and number of Services. . . . 98

List of Tables

3.1 Details of the IBM dataset. 35
3.2 Details of the SAP dataset. 35
3.3 Characteristics of generated small and large SBP graphs. 48

4.1 Characteristics of data inputs of generated SBP graphs and AM components. 72

5.1 Characteristics of data inputs of Cloud resources. 97

Chapter 1

Introduction

Contents

1.1 General Context 1
1.2 Motivation and Problem Statement 2
1.3 Objectives and Contributions 3
1.3.1 AMs Optimization in SBPs 3
1.3.2 AMs Components Optimization in Timed SBPs 4
1.3.3 Placement of AMs for the Management of SBPs in the Cloud . . . 4
1.4 Thesis Outline 4

1.1 General Context

Over the last decade, Cloud computing has appeared as an emerging paradigm, based on the *pay-per-use* economic model, for the provisioning of on-demand computing resources (*e.g.* networks, servers, storage, applications, and services) as a service over the Internet. These resources can be dynamically and easily provisioned and released with a minimal effort [10]. The Cloud paradigm basically utilizes three delivery service models known as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The adoption of this paradigm is rapidly increasing due to its capacity to reduce the management complexity and cost for enterprise customers while increasing scalability and flexibility and to provide ongoing revenue for providers [11]. According to a survey conducted by the IDG Enterprise [12] in 2016, 70 percent of the companies were at that time adopting the Cloud. However, 16 percent of them planned to adopt it in the following 12 months, while the remaining will have migrated to the Cloud within the following three years.

Cloud environments are increasingly used for hosting and executing business processes, especially Service-based Business Processes (SBPs) [13]. An SBP is built according to

Service Oriented Architecture (SOA) and consists of a set of elementary and heterogeneous services which are related in terms of their contribution to the realization of the business process. Assembling services into an SBP can be ensured by using any appropriate service composition specification, such as Business Process Model and Notation (BPMN) [14], Unified Modeling Language (UML) activity diagram [15].

However, executing SBPs in the Cloud is exposed to dynamic evolution workload and fluctuation during their life-cycle due to the dynamic nature of the environment. Consequently, the management of SBPs in Cloud environments is a challenging task. Indeed, the management of such business processes is time and effort consuming, not to mention the risks of being an error-prone process and the need for costly experts. In response to this problem, one of the main adapted technique is to resort to autonomic computing to enforce the autonomy and ensure the required Quality of Service (QoS) of SBPs. Autonomic computing [3] is the ability of a computing system to automatically and dynamically manage itself to respond to the requirements of the business based on on service level agreement. We advocate that coupling Cloud computing with autonomic management is really interesting since it allows intelligently coping with the dynamic evolution of Cloud environments with minimal human intervention [4]. This coupling is possible by means of Autonomic Managers (AMs) [3] that consist in collecting monitoring data from a running SBP, analyzing them, and planning and executing adaptation actions to meet the requirements of the managed SBP.

The autonomic management of an SBP implies the usage of one or more AMs, which are dedicated to the management of the services that make up the SBP. These services are generally deployed on different heterogeneous Cloud resources (*i.e.* virtual machines). Consequently, the efficiency of the management of SBPs as well as their performance depend highly on the allocation decisions of AMs and Cloud resources by these AMs in respect of the agreement between the service consumer and its provider.

1.2 Motivation and Problem Statement

When applied to SBPs in a Cloud environment, autonomic management becomes a complicated problem that faces many challenges. The first challenge faced by Cloud providers is to optimize the number of AMs for the management of SBPs. In fact, using a centralized AM for the monitoring and adaptation of a whole SBP can lead to management bottlenecks and single points of failure [16]. Thus, to reduce the load on the centralized AM and avoid bottlenecks and single points of failure, a solution is to dedicate different AMs to the SBP services, but this solution is resource-consuming. Consequently, it is interesting to autonomically manage SBPs such that the number of AMs is minimized while avoiding management bottlenecks.

It is obvious that determining the appropriate number of AMs for the management of SBPs is not sufficient to provide efficient management of deployed business processes. In fact, it is also necessary to consider the allocation decisions of Cloud resources to these AMs in the Cloud, which allows fulfilling the business process QoS requirements. Thus, the

second challenge is to optimize the allocation of Cloud resources to host and execute AMs that will be used to manage SBPs. In fact, due to the heterogeneity of Cloud resources and the diversity of the SBP services QoS needs, the allocation of Cloud resources to SBPs may result in high computing costs and an increase in the communication overheads. Consequently, it is interesting to select the right Cloud resources to achieve an efficient SBPs deployment such that the overall deployment cost is minimized while fulfilling SBPs QoS requirements.

Many attempts to provide autonomic management of applications in the Cloud have been proposed, but as we will explain, almost all the proposed solutions are limited to modeling and implementing autonomic mechanisms. It is worth noting that the existing approaches either do not take care of management cost issue, they do not optimize the Cloud resource allocation during the SBPs management process or do not include an efficient management of SBPs. In addition, the existing works on the optimal resource allocation are not suitable to solve the previously described problems because the number of these resources is considered as an input, which is not the case for AMs.

1.3 Objectives and Contributions

The main goal of this thesis is to propose an efficient solution for allocating Cloud resources to autonomic SBPs based on optimization programs that aim to provide an optimal number of AMs for the management of SBPs as well as optimal placement of these AMs in the Cloud. The objective of the optimization programs is to minimize the Cloud resource allocation cost while ensuring the QoS of deployed SBPs and avoiding the management bottlenecks. This thesis does not discuss all the aspects related to the autonomic management of SBPs. For example, we do not deal with modeling and implementing autonomic mechanisms or coordinating AMs. While we believe that a lot of issues related to autonomic management are important to address, the problem discussed here is complex enough to deserve a separate treatment.

To achieve our targeted objectives, this thesis makes the following contributions:

- AMs optimization in SBPs.
- AMs components optimization in timed SBPs.
- Placement of AMs for the management of SBPs in the Cloud.

1.3.1 AMs Optimization in SBPs

Executing SBPs in Cloud environments requires AMs to cope with the changes in these environments. However, using a centralized AM for managing a large number of services can lead to management bottlenecks and performance degradation of SBPs. While using an AM per service is resource-consuming, we argue for a decentralized approach that aims to reduce the number of AMs while avoiding management bottlenecks.

In our work, we opt for the use of directed graphs to formally represent SBPs to focus on all service dependencies between them when processing their management. We propose an exact optimization model that aim to find the optimal number of AMs for the management of SBPs. The amount of time needed to solve this problem grows exponentially with the size of the graph, so we propose near-optimal approaches that provide good solutions in a reasonable time. We use the exact optimization model for smaller graphs to benchmark our approaches that exhibit better scaling behavior, possibly at the expense of optimality.

1.3.2 AMs Components Optimization in Timed SBPs

An AM consists of four main components: a monitor, an analyzer, a planner, and an executor. Therefore, to further optimize the Cloud resource allocation cost, we propose to extend our work by considering these components separately in the optimization process. Doing so, we first propose an exact optimization model to find the optimal number of monitors, analyzers, planners, and executors for the management of SBPs such that their QoS is satisfied while avoiding management bottlenecks. Then we introduce a near-optimal algorithm that provides good solutions faster. We use the exact optimization model for smaller graphs to compare our algorithm to the optimal solution.

1.3.3 Placement of AMs for the Management of SBPs in the Cloud

Due to the heterogeneity of Cloud resources and the diversity of SBPs QoS needs, the allocation of Cloud resources to autonomic SBPs may result in high computing costs and an increase in the communication overheads and/or lower QoS if resource allocation is not well addressed. Indeed, several works [17, 18, 19] have been proposed to determine the optimal placement of SBP services in the Cloud, but there is no approach that tries to solve the optimal placement of AMs that will be used to manage these SBPs in the Cloud. Hence, we first suggest exact optimization models to solve this problem where the number of AMs is known or not known in advance. After that, we propose two different approaches that provide effective placement of AMs in the Cloud such that the required QoS of deployed SBPs is met while reducing the used Cloud resources in a cost-efficient way as per the economic model of the Cloud.

1.4 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 presents the different concepts used throughout our thesis as well as the state of the art of our research. In this chapter, we start by giving the definitions of Cloud computing, SOA and SBPs, autonomic computing as well as an overview of optimization problems. We then give an overview of the existing works related to autonomic computing as well as optimal resource allocation. Therein, we

will cite the different works in each area to highlight the existing limitations and emphasize the significance of our work.

Chapters 3, 4, and 5 are the core of our thesis which elaborate our approach to optimize Cloud resource allocation for the management of SBPs. Illustrative examples and experiment results are provided in each chapter.

In Chapter 3, we present our solution to determine the optimal number of AMs for the management of SBPs. Our objective is to minimize the number of AMs while avoiding management bottlenecks. The proposal entails the Integer Linear Programming (ILP) solution as well as a set of algorithms to solve the problem based on the representation of SBPs (*i.e.* graph with or without cycles). The main content of this chapter was published in [20, 21].

In Chapter 4, we push further our work by proposing to determine the optimal number of AMs components separately for the management of SBPs. The objective is to minimize their number while avoiding management bottlenecks. Our proposal entails the ILP solution as well as a set of algorithms to solve this problem. The main content of this chapter was published in [22, 23].

In Chapter 5, we present our solution to determine the optimal placement decisions of AMs or AMs components that will be used to manage SBPs. The objective is to minimize the Cloud resource allocation while fulfilling the QoS requirements of SBPs. The chapter content was published in [22, 24].

Finally, in Chapter 6, we conclude this thesis by summarizing our contributions and giving an outlook of future work.

Chapter 2

Background & State of the Art

Contents

2.1 Introduction 6
2.2 Background 7
2.2.1 Cloud Computing 7
2.2.2 Service-Oriented Architecture and Service-based Business Processes 9
2.2.3 Autonomic Computing 10
2.2.4 Optimization Problems 11
2.3 State of the Art 13
2.3.1 Evaluation Criteria 13
2.3.2 Autonomic Computing 13
2.3.3 Optimal Resource Allocation 18
2.4 Conclusion 23

2.1 Introduction

In this thesis, we aim to provide a solution for optimizing autonomic resources for the management of SBPs in Cloud environments. In order to understand the rest of this manuscript, we dedicate the first section of this chapter to briefly introduce a basic knowledge on different concepts and paradigms related to our work (Section 2.2). We start by presenting the environment of our work which is Cloud Computing. Afterwards, we define the service oriented architecture as well as SBPs that represent the type of applications that we basically target. Next, we introduce the concept of autonomic computing as well as a brief background about optimization problems. In the second section, we discuss the existing works on autonomic management in distributed systems and optimal resource

allocation to service-based applications (Section 2.3). Finally, we conclude the chapter in Section 2.4.

2.2 Background

In this section, we present definitions and basic concepts related to our work. First, we introduce the context of our work, which is Cloud Computing. Then, we introduce SBPs that represent the specific type of applications that we target. Finally, we define the concept of autonomic computing that we aim to address in our work.

2.2.1 Cloud Computing

Cloud Computing is an emerging paradigm in information technology. According to the National Institute of Standards and Technology (NIST) [10], it 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) which can be rapidly provisioned and released with minimal management effort or service provider interaction. Cloud computing is characterized by its pay-per-use economic model that allows a user to consume Cloud resources as needed. Virtualization forms the foundation of Cloud computing, as it is a key technology that makes Cloud computing possible. It abstracts physical hardware resources typically as Virtual Machines (VMs) with associate storage and networking connectivity, allowing applications to run in different VMs in a flexible manner [25]. Through virtualization, a Cloud provider can ensure the Quality of Service (QoS) delivered to users while achieving an energy efficiency and high server utilization.

Services in the Cloud are basically delivered under three layers:

- **Infrastructure as a Service (IaaS):** It provides to consumers processing, networks, storage, and other computing resources so as to deploy and run software, which can include operating systems and applications. Examples of IaaS infrastructures are Amazon AWS[1] and Google Compute Engine[2];

- **Platform as a Service (PaaS):** It provides to consumers appropriate resources to develop, deploy and manage applications onto the Cloud infrastructure using libraries, programming languages and tools supported by the Cloud provider. Examples of PaaS platforms are Cloud Foundry[3] and Windows Azure[4];

- **Software as a Service (SaaS):** The consumer is able to use applications running

[1] aws.amazon.com/fr/
[2] cloud.google.com/compute
[3] www.cloudfoundry.org
[4] azure.microsoft.com

on a Cloud infrastructure over the Internet. Examples of SaaS are Salesforce[5] and Netsuite[6].

Nowadays, more services have appeared, called as XaaS, where X stands for anything that can be provisioned and abstracted as services. Examples include NaaS for Network as a Service, DaaS for Desktop as a Service, etc.

Clouds can be provisioned following different models according to the requirements of users. Whenever the Cloud infrastructure is exclusively used by a single organization that owns, manages and maintains the Cloud, we talk about Private Cloud. However, if the Cloud infrastructure is used by a specific community of consumers from different organizations that own and operate it, we talk about Community Cloud. Otherwise, if the Cloud infrastructure is exposed to public use, we talk about Public Cloud. There is another model in which the Cloud infrastructure is composed of two or more distinct Cloud infrastructures (public, private, or community), we talk herein about Hybrid Cloud.

Figure 2.1 gives an overview of the introduced Cloud computing concepts.

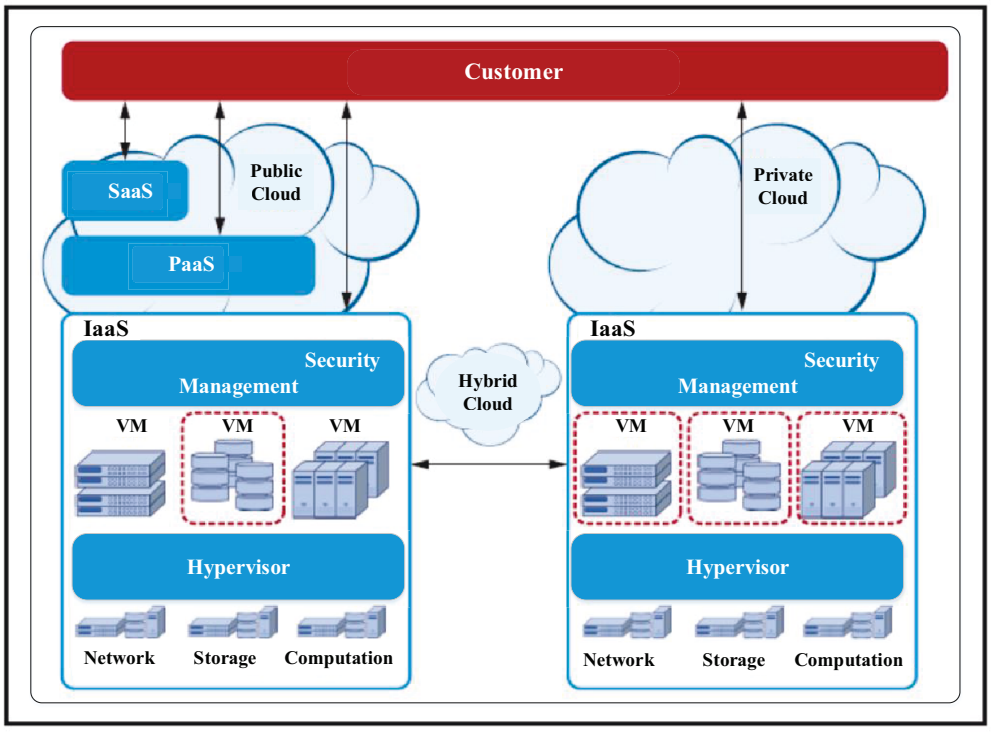


Figure 2.1: Cloud computing models [1].

[5] www.salesforce.com
[6] http://www.netsuite.com/portal/home.shtml

2.2.2 Service-Oriented Architecture and Service-based Business Processes

Service-Oriented Architecture (SOA) is a software architecture that is based on services as fundamental elements for developing and integrating applications [26]. A service is a self-describing component that supports rapid, low cost development and deployment of distributed applications. The goal of this architecture style is to meet the requirements of loosely coupled, standards-based, and protocol-independent distributed computing. As defined by Papazoglou [27], *SOA is a logical way of designing a software system to provide services to either end user applications or other services distributed in a network through published and discoverable interfaces.*

The SOA is structured around the three actors illustrated in Figure 2.2: a service provider, a service consumer, and a service registry; whereas, the interactions between these actors involve publish, find and bind operations. Service provider is the role of a software entity providing a service. Service consumer is the role assumed by a requesting entity that seeks to consume a specific service. However, service registry is the role assumed by an entity that maintains information on available services as well as the way to access them.

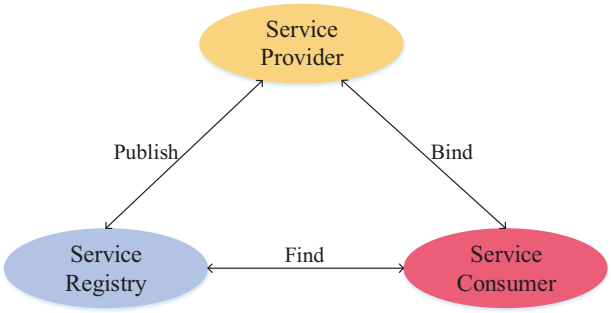


Figure 2.2: SOA actors and roles.

The advantage of this approach lies in the loose coupling of the services composing an application. Services are provided by independent parts of a platform, which implies that a client using any programming language, operating system and any computational platform can use the services, although different service providers and service customers may use different technologies to implement and access the services. Thus, an application can be made up of different services provided by heterogeneous parts with respect to their services descriptions.

A business process is an ordered set of services aiming to reach a business objective or policy goal, normally within the context of an organizational structure that defines functional roles and relationships. A process may be wholly contained in a single organizational unit or may span mutiple different organizations such as in a customer-supplier relationship [28].

An SBP is a business process that consists in assembling a set of elementary IT-

enabled services which are related in terms of their contribution to the overall realization of the process. A service is typically the smallest unit of work that represents a module offering computation or data capabilities. It carries out the business activities of an SBP. Assembling services into an SBP can be ensured using any appropriate service composition specification such as Event-driven Process Chains (EPCs) [29], Business Process Modeling Notation (BPMN) [14], and UML activity diagram [15]. In this thesis, we aim to abstract away from these specific notations and focus on basic commonalities of these languages [30]. Accordingly, we define an SBP as a set of services and gateways that are connected by control-flow arcs. Throughout this thesis, we consider the following types of gateways; each acts as either a split or a join node. Split gateways have exactly one incoming edge and at least two outgoing edges. Join gateways have at least two incoming edges and exactly one outgoing edge.

- **Parallel (AND)** indicates that all branches must be executed in parallel. AND-split enables all its outgoing branches to execute concurrently. AND-join waits until all its incoming branches have completed their execution.

- **Inclusive (OR)** is used when at least one branch must be executed. OR-split enables one or more of its outgoing branches to execute whose condition evaluates to true. OR-join waits until all the latter branches have completed their execution.

- **Exclusive (XOR)** specifies that only one branch must be executed. XOR-split enables only one of its outgoing branches to execute. XOR-join waits until the chosen branch has completed its execution.

2.2.3 Autonomic Computing

Autonomic computing aims to build computing systems capable of self-management without needing human intervention. It is inspired by the human body's autonomic nervous system that monitors heartbeat, checks blood sugar levels, and keeps the body temperature normal without any conscious effort from the human [2]. The term autonmic was originally coined by IBM in 2001 [3]. IBM defines autonomic computing as the ability to manage computing resources that automatically and dynamically respond to the requirements of the business based on Service Level Agreement (SLA).

Management tasks like monitoring, protection, configuration and optimization are the non-functional objective of most applications, but they have a critical importance for applications to accomplish their tasks. Thus, the challenge is to enable self-managing computing systems that take control of all the non-functional tasks in order to let the developers focus on the main functional tasks of applications. To do so, there must exist a way to develop self-governing features and to integrate them with applications.

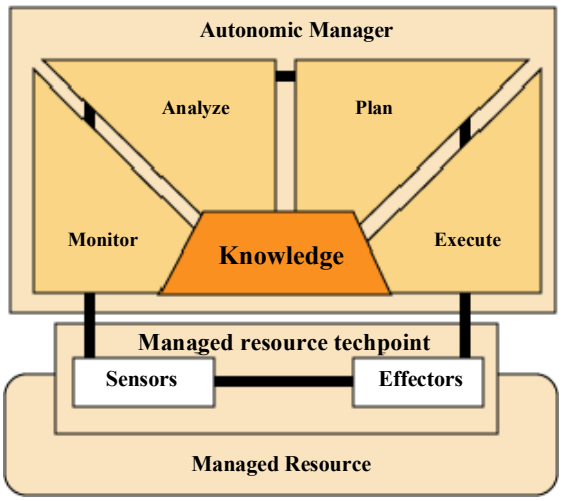


Figure 2.3: IBM architecture of autonomic resource [2].

To achieve autonomic computing, IBM has suggested a reference model for an autonomic resource called Autonomic Manager (AM), also known as the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) loop [2], as illustrated in Figure 2.3. An AM is a software agent that implements the autonomic behavior. A managed resource represents any hardware or software resource that is coupled with an AM to exhibit an autonomic behavior. Sensors are used by an AM to collect information about the state of a managed resource, while effectors are used by an AM to carry out changes to a managed resource. A common element, called Knowledge, represents the management data (*e.g.* adaptation strategies, change plans, etc.), which can be shared between the phases (*i.e.* monitor, analyze, plan, and execute) of the control loop. An AM gathers monitoring information from the managed resource through sensors where this information is stored in the knowledge base. The latter data are then analyzed in order to determine whether an adaptation is required. If it is the case, an adaptation plan is provided by the AM and the latter plan is executed over the managed resource through effectors.

2.2.4 Optimization Problems

2.2.4.1 Classification of optimization problems

Optimization problems can be classified according to several criteria. An important classification of these problems is based on the nature of equations for the objective function and the constraints. Optimization problems can be classified into four classes: linear, nonlinear, quadratic, and geometric programming[31].

- **Linear Programming (LP):** In this type of problems, both the objective function and the constraints are linear functions of variables. LP problems can be classified

according to their decision variables as Integer Linear Programming (ILP), where all variables are integer, and Mixed Integer Linear Programming (MILP) problems, where some but not all variables are integer.

- **Nonlinear Programming (NLP):** In this type of problems, any of the functions among the objective function and the constraints is nonlinear.
- **Quadratic Programming (QP):** This category of problems has a quadratic objective function and linear constraints.
- **Geometric Programming (GP):** In this type of problems, the objective function and the constraints are expressed as polynomials.

Optimization problems can be classified also based on the deterministic nature of the variables as deterministic and stochastic programming problems [31].

- **Deterministic Programming:** In this type of optimization problems, all the design variables are deterministic. A model is deterministic if, for a given input, it always produces the same output.
- **Stochastic Programming:** In this type of problems, some or all of the parameters of the optimization problem are considered as random or probabilistic variables (nondeterministic or stochastic). A stochastic variable is a variable that has different values with certain probabilities.

2.2.4.2 Complexity of optimization problems

There are mainly two classes of optimization problems: *P* and *NP* [32]. *P* is the class of problems solvable by algorithms operating within a polynomial time (if it is run $O(n^k)$ steps where *k* is a constant and *n* denotes the problem size). *NP* is the class of problems solvable by non-deterministic algorithms operating within a polynomial time. It stands for a non-deterministic polynomial.

Definition 2.2.1. *A decision problem is NP-Hard if every problem in NP can be reduced to it in polynomial time.*

Definition 2.2.2. *A decision problem is NP-Complete if it is in NP and is also NP-Hard.*

In fact, *NP-Hard* problems are generally difficult to solve and time-consuming. They are often addressed using approximation algorithms.

Definition 2.2.3. *An approximation algorithm is an algorithm that always returns a feasible solution in polynomial time to an optimization problem.*

Although approximation algorithms provide near-optimal solution, it is interesting to study the quality of approximate solutions where there are two equivalent measures of this quality: the relative error and the optimality gap.

Definition 2.2.4. *Given an optimization problem, for any instance i of this problem, we denote by m(i) the feasible solution and by opt(i) the optimal solution with respect to instance i. The relative error is defined as:*

$$E(i) = \frac{m(i) - opt(i)}{opt(i)}$$

For both minimization and maximization problems, if the relative error is equal to 0, the approximate solution is optimal. It becomes close to 1 when the obtained solution is very poor.

Definition 2.2.5. *The optimality gap of an approximate solution is expressed in % and is defined as:*

$$G = E(i) \times 100$$

2.3 State of the Art

In this section, we present a selection of works around autonomic computing in Cloud and distributed environments (Section 2.3.2) and optimal resource allocation (Section 2.3.3). In fact, a plethora of studies exists in the literature aiming to provide autonomic computing and optimal resource allocation solutions, but to the best of our knowledge, these proposals treat the two areas separately. In this manuscript, we will refrain from citing all these studies to highlight just a selection of them that we believe representative.

2.3.1 Evaluation Criteria

To address the autonomic management of SBPs issues mentioned in Section 1.2, the proposed approach should provide appropriate ways to efficiently manage SBPs by determining the appropriate number of AMs and selecting the appropriate Cloud resources to host and execute these AMs such that the management cost is minimized while avoiding management bottlenecks to fulfill the QoS requirements with minimal costs. In addition to that, it should cover complex business process patterns like AND-blocks, OR-blocks, XOR-blocks and Repeat Loop which are very common in real world business processes [30]. It should also be suitable for small and large SBPs and structured and unstructured SBPs.

2.3.2 Autonomic Computing

IBM is a pioneer in the field of autonomic computing that proposed a dedicated toolkit, which is a set of tools and technologies designed to permit users to develop autonomic behavior for their systems. The authors in [3] introduced the needed steps to add autonomic capabilities to resources. One of the basic tools is the autonomic management engine that includes representations of an AM which provides self-management properties

to managed resources. Furthermore, IBM suggested several tools to allow managed resources to create log messages using a standard format understandable by the AM. This is achieved using a touch-point that consists of a sensor and an effector. The sensor detects the state of the managed elements and generates logs, while the effector is used to carry out adaptations on the managed resource. Moreover, an adapter rule builder is proposed to create specific rules to generate adaptation plans. Figure 2.4 illustrates IBM's vision of the structure of the autonomic resources and their interactions.

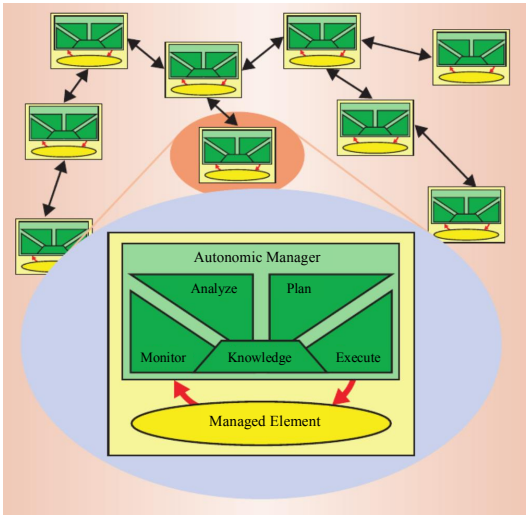


Figure 2.4: Structure of autonomic resources and their interactions [3].

In [4], the authors put forward a conceptual architecture enabled the autonomic management of SaaS applications in Cloud environments. The proposed architecture was basically composed of a SaaS Web application that hosted the SaaS application and negotiates the SLA between the service provider and consumers, an Autonomic Management System (AMS) integrated in the PaaS level, and an IaaS layer that provided Cloud resources. As shown in Figure 2.5, the AMS incorporates an application scheduler assigns each service in an application to Cloud resources and an energy-efficient scheduler that minimizes the energy consumption during the application scheduling process. The AMS implements logic for provisioning and managing virtual resources according to the resource requirements specified by the application scheduler. It incorporates also a component to detect security infraction and attack tentatives.

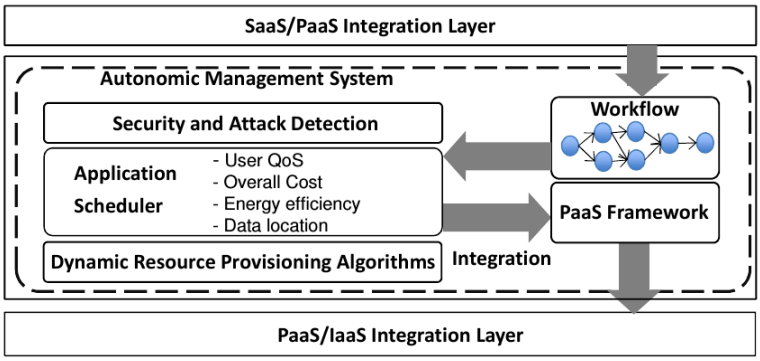


Figure 2.5: System architecture for autonomic Cloud management [4].

Rainbow [5] is a framework that uses an architecture-based approach to add self-adaptation mechanisms to software systems. The proposed framework uses an abstract model to represent an application as a graph. Vertices in the graph represent components, and arcs represent interactions between components. A model manager is used to continuously adapt the model. It collects monitoring data from the model through probes. The latter data is analyzed using a constraint evaluator to detect violations and trigger adaptation. The appropriate adaptation plan is chosen using an adaptation engine and is applied using an executor. As shown in Figure 2.6, the framework is divided into architecture, translation and system layers. The architecture layer consists of the needed components to self-manage a system. The system layer describes the managed system access interface (*e.g.* probes, effectors). Between these two basic layers, the translation infrastructure is used as a mediator to exchange information.

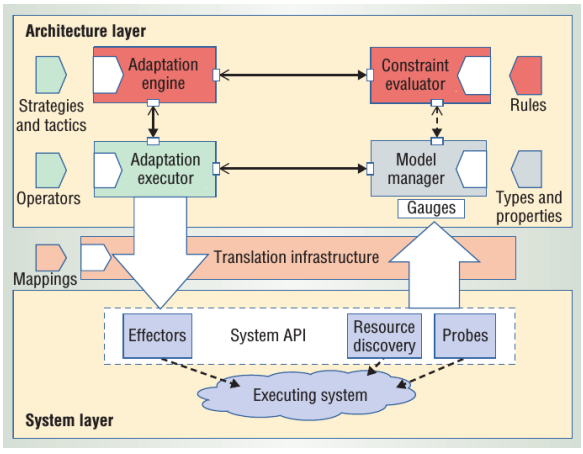


Figure 2.6: Rainbow framework [5].

In [33], the authors proposed a framework for dynamically adapting applications to the changes of environments. A real-time monitor checks applications and triggers adaptation

if a constraint violation occurs to meet the desired goals. The functionalities of the framework are implemented in separate components: sensors, actuators, distributed name servers, and clients. Sensors and actuators are used to remotely collect and modify the values of application variables. They have to register themselves to a name server (registry). Clients specify a set of desired properties, and the manager provides start points to the interesting sensors or actuators in order to allow clients to establish direct communication with applications.

The authors in [6] introduced a generic framework to create Java-based applications with self-managing properties. It was an open source project that used Java management extensions to enable self-managing capabilities where management logic was separate from application logic. The framework implemented adaptive behavior using policy languages or Java codes following condition-action rules. As depicted in Figure 2.7, it consists of two main elements: the Execution Engine and a set of Services. The Execution Engine enables AMs to perform their jobs using services provided by the Services element. Management logic is implemented as a set of entities called *processes* where each process may implement one or more consecutive components of an AM. The composition of processes forms an execution chain which acts as an AM.

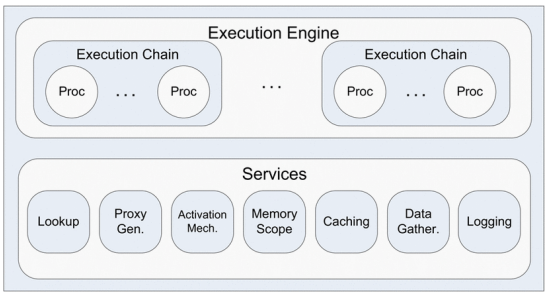


Figure 2.7: StarMX high-level static architecture [6].

C. Ruz et al. [7] suggested a framework that allowed the monitoring and management of component-based applications. As presented in Figure 2.8, the authors separated components for the monitoring, SLA analysis, decision, and execution of a classical AM. These components were attached to each component to manage it, where these components could be added or removed if necessary and each one could communicate with other components. The monitoring component would gather information from the managed component. The SLA analyzer component would compare the collected data to previously defined SLA. The decision component would generate a sequence of actions to be applied on the managed component to meet SLA. The execution component would apply the decided actions.

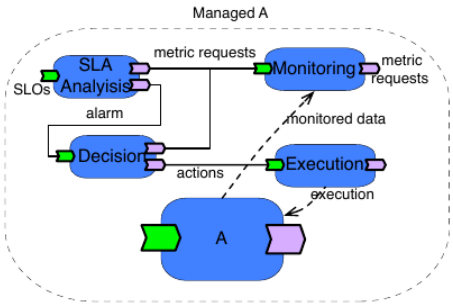


Figure 2.8: Autonomic component "A" [7].

N. Belhaj et al. [34] introduced an approach to improve the decision making process of a traditional AM to dynamically adapt component-based applications. To do so, the authors equipped the analyzer component with sophisticated learning blocks instead of using inflexible hand-coded strategies where its decision making process was formulated as a Markov decision process with a finite set of states and actions. During each state transition, a reinforcement signal would indicate to the proposed decision maker whether it would choose the appropriate action for the current state or not. In this work, each component of an application is self-managed by its own AM.

In [35], the authors focused on the collaboration between multiple policy-based AMs to efficiently manage the overall system. AMs are organized in a hierarchical structure where the higher-level AMs have more authority over AMs of lower levels. The lower-level AM is responsible for allocating resources, such as memory and CPU, to the Web server to improve its response time and avoid SLA violations. An AM requests the help of a higher level AM when no further local adjustments are possible. These AMs communicate by exchanging predefined messages by means of a message broker.

F. de Oliveira et al. [8] proposed a framework for the coordination of AMs in Cloud environments. This framework would improve the synergy between AMs. As shown in Figure 2.9, each application is managed by its own Application AM (AAM) which is responsible for determining the best architectural configuration and the minimum amount of VMs needed to deliver the best QoS under a certain workload. the IaaS layer (physical machines and VMs) is managed by a single Infrastructure AM (IAM) which is responsible for the optimal placement of VMs. The IAM is composed of a public knowledge while each AAM maintains a private knowledge. AAMs can make changes to the public knowledge using a synchronization protocol. This protocol uses a token to synchronize the actions executed by the access of AMs to the shared knowledge in order to avoid concurrency and consistency problems and to make the knowledge available to both SaaS and IaaS resources. An event-based coordination protocol is also proposed to coordinate the AMs actions.

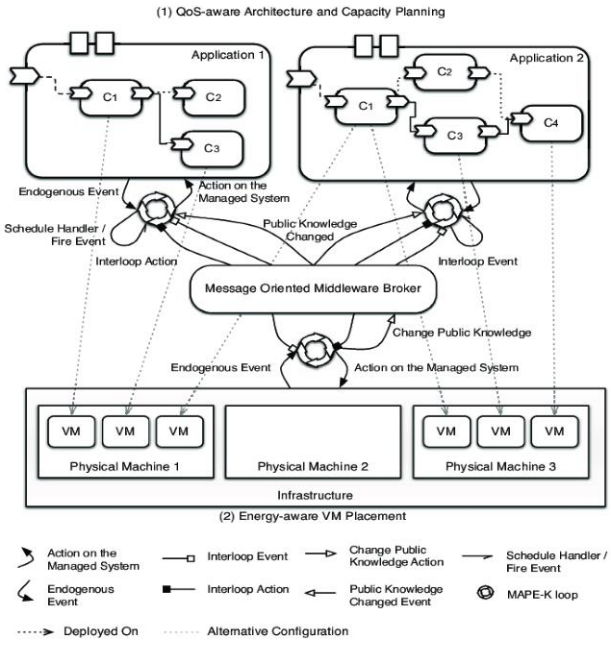


Figure 2.9: Architecture overview of self-adaptation framework [8].

Beside the abovementioned approaches, there is a lot of other work in the literature related to autonomic computing [36, 37, 38, 39, 40, 41, 42]. However, to the best of our knowledge, all the existing works have been limited to modeling and implementing autonomic environments. They either do not take care of the management cost issue by using an AM per SBP service, or do not include an efficient management of SBPs by using a centralized AM to manage all the SBP services that can lead to management bottlenecks and/or a violation of QoS. In addition, we have not found any work that aims to achieve good trade-offs between management performance and management costs by attempting to optimize the used AMs and Cloud resources allocated to these AMs during the management process while respecting QoS constraints and avoiding management bottlenecks. In contrast, in our proposal, we will give solutions to manage SBPs such that the number of AMs as well as the Cloud resources allocated to these AMs are optimized in a cost-efficient way while avoiding management bottlenecks and fulfilling the QoS constraints.

2.3.3 Optimal Resource Allocation

2.3.3.1 Cloud environments

In a Cloud environment, the resource allocation process consists in selecting an optimal set of physical machines to host the received services while respecting resource and QoS constraints. To solve this NP-Hard problem [43], many works have been proposed that

aim to achieve good trade-offs between solution quality and computation time. Some of them are summarized below.

In [44], the authors presented a cost-efficient deployment approach for Business Processes (BPs) in the Cloud. They proposed an optimization model based on the MILP technique to achieve an optimal scheduling and placement of services that would compose a BP on VMs. The objective was to minimize the cost of leased Cloud resources and the penalty cost that would arise when a process execution did not meet a user-defined deadline while guaranteeing the defined QoS. In [17], the authors extended this work by considering an inter-cloud data transfer cost.

In [45], Bessai et al. proposed resource allocation and scheduling models for BP applications in the Cloud. A BP was represented as a Directed Acyclic Graph (DAG). The authors defined two optimization models to determine the optimal assignment of tasks that would make up a BP to VMs such that the overall execution time (respectively cost) was minimized. They took into account the execution time and cost for each task and the data transfer time and cost between tasks that depended on resources that executed them. They also proposed three optimization algorithms that provide near-optimal solutions based respectively on the execution cost, the overall execution time, and a priority-based Pareto efficient combination.

Fakhfakh et al. [46] suggested a provisioning solution of Cloud resources for dynamic workflow applications. The objective was to find a Cloud resource allocation that would minimize the execution cost while satisfying a defined deadline. The authors put forward an algorithm that mapped Cloud resources (VMs) to workflow tasks, and if needed, some adaptation actions could be performed to cope with exceptional situations and evolution where a workflow was modeled as DAGs. They considered the data transfer time between tasks that only depended on the amount of data to be transferred between the corresponding tasks.

The work presented in [47] addressed the problem of resource allocation for BPs. To this end, the authors proposed an approach that would improve the process structure based on resource allocation requirements such that the overall cost was minimized while meeting the process execution time requirement. The structure of a BP was modeled as a DAG. They applied a basic allocation strategy to minimize the overall cost without considering the time limit. Then an adjustment strategy was applied to adjust the allocation scheme to ensure that the overall execution time of the process did not exceed the time limit.

In [48], the authors addressed the problem of scheduling and resource provisioning for scientific workflow where a scientific workflow was modeled as a DAG. They looked for an efficient mapping of tasks to available resources and selecting of the best resource provisioning plan under budget and deadline constraints. To this end, they proposed a set of algorithms based on offline and online strategies for task scheduling and resource provisioning that relied on estimates of task runtimes.

Goettelmann et al. in [49] suggested an algorithm for deploying BPs on different Cloud platforms under security constraints. The main idea was to split a BP into sub-processes and deploy them in different Clouds to meet security requirements. A security level was

assigned to each BP task describing its security requirements, and arbitrary security levels were assigned to Cloud providers. The algorithm would select the suitable Cloud providers which offered the minimum communication costs between the derived sub-processes.

Beside the aforementioned approaches, there have been many other attempts to optimize resource allocation for deploying BPs in the Cloud [50, 51, 52, 53, 54, 55, 56]. Up to our knowledge, almost all of the proposed solutions have not dealt with the repeat-loop pattern, i.e. SBPs that could be represented as directed graphs with cycles. In addition, the existing works have not tackled the XOR-block pattern (*i.e.* XOR-Split, XOR-Join) in the optimization process. Furthemore, such work has not dealt with unstructured SBPs, and some of these studies [19, 51] proposed exact solutions that can only solve for small SBPs. Moreover, the most important was that these works took as inputs the services that composed an SBP and the dependency relationship between these services and selected an optimal set of Cloud resources to host and execute these services while meeting resource and QoS constraints. In our case, the number of AMs and the dependency relationship between AMs and services are not known in advance. In fact, even though we consider an AM as a service, the existing works cannot be adapted to address our research needs. However, in this manuscript we propose solutions that deal with the different business process patterns (*i.e.* AND-blocks, OR-blocks, XOR-blocks and a repeat-loop) and can be applied to small and large SBPs as well as structured and unstructured SBPs.

2.3.3.2 Graph theory techniques

Besides the aforementioned approaches, there has been a lot of work on solving optimization problems based on graph theory techniques. In fact, graph coloring, graph partitioning and shortest and longest paths are among the most important and fundamental techniques that have been widely applied in the literature to find the solutions for real world problems in optimization, such as network design and computer science, and we can adopt/adapt them for solving the optimization problems that we tackle in this thesis. Broadly speaking, these problems belong to the class of NP-hard problems [57], and generally only small problem sizes can be solved in acceptable times. Therefore, based on these problems, several heuristic approaches have been proposed to solve these various problems. Some of these studies are summarized in the following.

The graph coloring problem involves mapping a minimum number of colors to the vertices of a graph such that each two adjacent vertices, i.e two vertices connected with an edge, do not have the same color. Graph coloring has a wide range of applications, including task scheduling, resource allocation, and frequency assignment. In [58], the authors proposed a new graph coloring model for resource reservation in Cloud datacenters with minimum datacenters energy consumption and maximum resource utilization (VMs). They introduced an ILP formulation to define the graph coloring optimization problem. They also suggested a near-optimal heuristic that would provide the efficient reservation of VMs in acceptable computational time even for large problem sizes. The main idea was to build a graph where vertices represented requested resources that would be associated

(colored) to VMs, and links represented time overlap between these requested resources. It was noted that each VM represented a unique color in the graph coloring problem. With this representation, the VM reservation consisted in determining an optimal mapping between the requested resources and VMs.

In [59], the authors studied the problem of service deployment for efficient execution of SaaS applications in the Cloud. Among various QoS aspects of applications, they focused on the response time of applications. The authors proposed deployment strategies to determine the placement of services on VMs based on optimizing inter-task communication costs and parallelism. In order to reduce communication costs and increase parallelism simultaneously, two kinds of graphs, which represented two types of relationships among services, were proposed. The first one, namely the Service Dependency Graph (SDG), described the interdependence relationship between services, whereas the second one was the Service Concurrence Graph (SCG) where an edge between two vertices would indicate that the two corresponding services could run concurrently. To raise parallelism, the service deployment problem was transformed into a graph coloring problem for SCG where the number of colors represented the available VMs in order to deploy concurreny services onto different VMs for increasing the potential parallelism and thus improving the execution performance.

In [60], the authors put forward an algorithm for the radio frequency assignment problem. The objective was to minimize the number of radio frequencies used by transmitters at different locations while avoiding interference. The problem was closely related to graph coloring where vertices represented transmitters and edges represented possible interferences.

Graph Partitioning is the problem of dividing a graph into a given number of sub-graphs such that the number of edges connecting these sub-graphs is minimized. It has been widely used in many real life applications such as problems involving load balancing in distributed computing, parallel processing, and clustering in data mining. In [61], the authors addressed the problem of social network data placement and replication in Cloud datacentres with a minimum monetary cost while satisfying the latency requirement for users to access their own data and their friends' data in an acceptable time. They introduced a mathematical formulation to solve this optimization problem and proposed a graph-partitioning based algorithm that would divide a social graph of users into different connected partitions where the vertices represented social network users and the edges represented the list of friends for these users. The algorithm started with users with the biggest number of friends. These users and their friends were assigned to random partitions. Then for all unassigned users, it started with user's neighbours, and all their friends were assigned to the dominant partition between their neighbours. After that, for each partition, data items related to all users were placed in the nearest datacenter to the user who had the most number of friends in this partition. Data were replicated on multiple datacenters until the required latency was satisfied.

In [62], the authors addressed the problem of optimal deployment of software applications in the Cloud. They introduced an ILP formulation to define the problem and

proposed a heuristic approach that was based on a graph partitioning algorithm. The algorithm would partition the components that composed a software application on a set of interconnected machines in the Cloud such that the communication cost between the components was minimized. The vertices of the graph represented components that made up a distributed software application, and the edges represented communication overhead between those components.

In [63], the authors focused on the problem of workflows scheduling such that the communication cost was minimized. They proposed a graph partitioning approach to partition tasks that composed a workflow over a given number of computation nodes while minimizing inter-node communication.

Shortest and longest paths are the problems of finding a path between two vertices in a graph such that the sum of the weights of its constituent edges is, respectively, minimized and maximized. They are widely used in road navigation and network routing. In [64], the authors suggested an algorithm based on partial critical paths for workflow scheduling in utility grids. The objective of the proposed algorithm was to minimize the total workflow execution cost while satisfying a user-defined deadline. The algorithm consisted of two main phases: deadline distribution and planning. In the first phase, it tried to assign sub-deadlines to all tasks of the critical path of the workflow which was the longest execution path in the workflow [65], such that it could complete before the user's deadline and its execution cost was minimized. Then it determined the partial critical path to each assigned task, and it recursively assigned sub-deadlines to the tasks of the partial critical paths. In the planning phase, the algorithm selected the cheapest service for each task while meeting its sub-deadline.

In [66], the authors addressed the end-to-end QoS problem for composite complex BPs. The objective was to maximize the user benefits and minimize the cost while meeting the QoS needs. In fact, every service provider would offer many service levels for a service functionality. The authors put forward an approach that consisted in modeling the problem as a constrained shortest path problem, where vertices represented the service levels of services. If service i is connected to service j, then all service levels in i are connected to all service levels in j. The algorithm would connect all vertexes that had no incoming edges to a source node and all vertexes that had no outgoing edges to a destination node. For each edge, it added a cost, a benefit, and a delay. Then it determined the path with the highest utility between source and destination nodes based on the shortest path technique to select the most suitable services and service levels for clients and construct optimal BPs.

There are many other approaches to solve optimization problems based on graph theory techniques [67, 68, 69, 70, 71]. To the best of our knowledge, all the existing proposals have not dealt with the semantics of the nodes of a graph (*e.g.* XOR-Split, XOR-Join) in the optimization method. Moreover, unlike these studies, the number of AMs and the dependency relationship between AMs and services are not known in advance. However, in this manuscript we propose solutions that deal with the semantics of the different BP patterns.

2.4 Conclusion

In the first part of this chapter, we presented the basic concepts related to the thesis. We introduced Cloud Computing as the environment of our research. Afterwards, we defined the SOA and specified the type of applications that we target, namely SBPs. Next, we presented the autonomic computing concept as well as a brief background about optimization problems. In the second part of this chapter, we gave an overview of the existing works on autonomic computing, optimal Cloud resource allocation to SBPs and optimization based on graph theory, and we presented the difference between existing approaches and our approach.

We start presenting in detail our approaches in the next chapters. We will propose an approach for determining an appropriate number of AMs in SBPs (Chapter 3). In Chapter 4, we will propose an approach for determining an appropriate number of AMs components in timed SBPs. Thereafter, we will propose an approach that aims to optimally allocate Cloud resources to AMs for the management of SBPs in Chapter 5.

Chapter 3

Autonomic Managers Optimization in SBPs

Contents

3.1 Introduction 24
3.2 Problem Description and Formulation 25
3.2.1 SBP Modeling 25
3.2.2 Problem Statement 26
3.2.3 Problem Formulation 27
3.2.4 Need for Approximate Approach 29
3.3 AMs Optimization in SBP Graphs with Cycles 29
3.3.1 Proposed Approach 29
3.3.2 Illustrative Example 33
3.3.3 Performance Evaluation 34
3.4 AMs Optimization in SBP Graphs Without Cycles 38
3.4.1 Proposed Approach 38
3.4.2 Illustrative Example 41
3.4.3 Performance Evaluation 44
3.5 Conclusion 50

3.1 Introduction

Executing SBPs in the Cloud requires autonomic management to cope with the dynamism and scalability of Cloud environments. It is obvious that using one centralized Autonomic Manager (AM) for the monitoring and adaptation of a large number of distributed services may cause a performance degradation and/or bottlenecks in the SBP management. It

is also obvious that using one AM per service may result in a high management cost. Therefore, we aim in this chapter to find the optimal number of AMs for the management of SBPs in order to reduce the management cost while avoiding management bottlenecks.

This chapter is organized as follows: First, we describe the problem of finding the minimum number of AMs for the management of SBPs while avoiding management bottlenecks, and we formulate it as an optimization model in Section 3.2. Then we present our defined algorithms, where SBPs can be represented as graphs with cycles (respectively without cycles) in Section 3.3 (respectively 3.4). Illustrative examples as well as experiments results are provided for both cases. Finally, we conclude the chapter in Section 3.5.

3.2 Problem Description and Formulation

In this section, we first show how to map an SBP to a directed graph, and we present some preliminary notions on business processes. Then we define the problem that we tackle in this chapter, followed by an Integer Linear Programming (ILP) formulation to solve it. Finally, we introduce the need for an approximate approach.

3.2.1 SBP Modeling

An SBP is a collection of related services, gateways and possibly events that accomplish a specific goal. Many graphical notations are available to model SBPs such as EPC [29], UML Activity Diagram [15], and BPMN [14]. In this thesis, we try to abstract as much as possible from these specific notations. Therefore, as an SBP model can be mapped to a graph, we use graph theory to represent it as a directed graph called *SBP graph* according to the following definition, which is inspired from the BP graph definition given in [72].

Definition 3.2.1 (SBP graph)**.** *An SBP graph $G = (V, E, I, \tau, ID)$ is a directed graph where:*

- *V is the set of nodes. It represents the set of services, events, and gateways;*
- *$E \subseteq V \times V$ is the set of edges that represents the data dependencies between nodes, such that $(v_i, v_j) \in E$ if the output data of the node i is required for the execution of the node j;*
- *$I \subset V$ is the set of initial nodes;*
- *$\tau : V \leftarrow T$ is a function that assigns, for each node $v \in V$, a type $t \in T$. T={service, start event, end event, gateway}, where gateway={AND, OR, XOR};*
- *$ID : V \leftarrow \mathbb{N}$ is a function that assigns, for each node $v \in V$, a unique identifier $id \in \mathbb{N}$.*

An SBP can be represented as a structured SBP that consists of a set of nodes and transitions between them. A node can be a service, AND-split, AND-join, OR-split, OR-join, XOR-split, or XOR-join. In the following, we introduce the definition of a structured SBP.

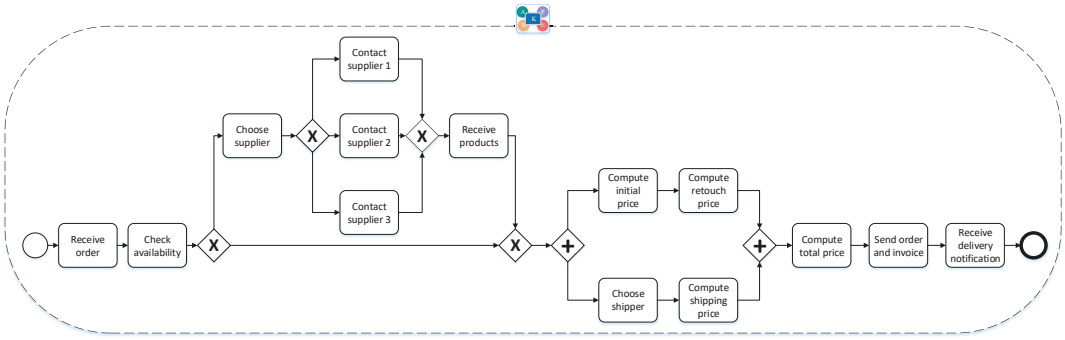
Definition 3.2.2 (Structured SBP)**.** *A structured SBP is inductively defined in [73] as follows:*

- *An SBP that consists of a single service is a structured SBP. This service is both initial and final.*
- *Let P_1 and P_2 be structured SBPs. Their concatenation is also a structured SBP, where the final node of P_1 has a transition to the initial node of P_2. The initial and final nodes of this SBP are the initial and final nodes of P_1 and P_2, respectively.*
- *Let P_1,...,P_n be structured SBPs, s an AND-split (respectively OR-split, XOR-split), and f an AND-join (respectively OR-join, XOR-join). An SBP that has s as an initial node and f as a final node as well as transitions between s and the initial node of P_i ($i \in \{1, 2, ..., n\}$) and between the final node of P_i and f is also a structured SBP. The initial and final nodes of this structured SBP are s and f, respectively.*

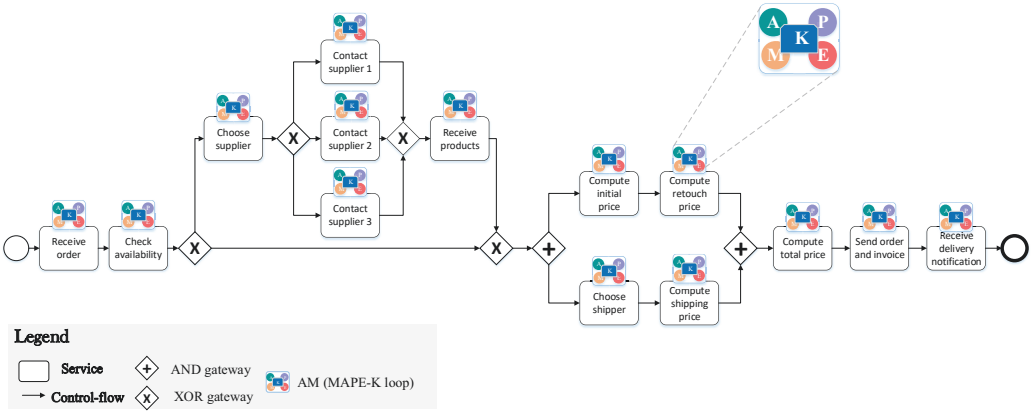
3.2.2 Problem Statement

Determining the optimal number of AMs for the management of a given SBP is a crucial issue. Figures 3.1a and 3.1b illustrate two naive approaches that represent two different extremes to this problem. In the first approach, a centralized AM is used to manage all the services that make up SBP [8]. This can lead to management bottlenecks and performance problems due to the large amount of data that can be periodically generated by multiple services at the same time and will be processed by a single AM. In the second approach, each SBP service is managed by its own AM [74]. Although this approach avoids management bottlenecks, it incurs high management costs in terms of number of used AMs. Therefore, it is interesting to minimize the number of AMs that will be used to manage SBPs while avoiding management bottlenecks.

The main question we ask is: How to reach a trade-off between these two extremes?



(a) Management scenario using one centralized AM for monitoring and adaptation of the whole SBP



(b) Management scenario using AM per service to monitor and adapt it

Figure 3.1: First research problem: Finding appropriate number of AMs for management of SBP services.

3.2.3 Problem Formulation

We start this section by introducing some assumptions and notations in order to facilitate the formulation. Possible management bottlenecks can be caused by the number of AMs, their placement, the communications between them either directly or through a common knowledge base, and other properties such as security and privacy mechanisms. We tackle in this chapter the optimization of the number of AMs to address bottlenecks of the management of SBPs. Without loss of generality, we assume that all AMs have the same requirements in terms of hardware resources (RAM, CPU, etc.), so that the optimization of their hosting cost is reduced to the optimization of their number.

Given an SBP graph, we denote by:

- **S** the set of services;

- **M** the set of candidate AMs that may be assigned to *S* ($|M| = |S|$, where $|M|$ and $|S|$ are, respectively, the cardinality of *M* and *S*). In fact, in the worst case, different AMs will be assigned to *S*;

- **P** the set of sets of services where an element of *P* is a pair of services that are at the same level in the SBP graph and their parent node is an AND/OR -Split. In order to determine *P*, we implement an algorithm with a polynomial time complexity that first determines the sets of services that belong to the same level in the graph, and then it checks for each pair of services that are at the same set whether their parent node is an AND/OR -Split. If it is the case, this pair is added to *P*.

  Consider the example illustrated in Figure 3.1b, *P*={(Compute initial price, Choose shipper),(Compute retouch price, Compute shipper price)}.

Towards this end, the following decision variables are defined:

- x_m^s takes 1 if AM $m \in \mathcal{M}$ is assigned to service $s \in \mathcal{S}$, and 0 otherwise;

- y_m takes 1 if AM $m \in \mathcal{M}$ is used by at least one service, and 0 otherwise.

Given the above assumption and notations, our problem can be formulated in equations [3.1-3.6] where objective function 3.1 aims to minimize the number of AMs for the management of an SBP. Constraint 3.2 makes sure that each service is managed by only one AM all along the SBP life cycle. The authors in [75, 16] adopted the principle of using multiple AMs for the management of applications in order to avoid management bottlenecks, but they did not provide any means to optimize their number. In our work, an AM is able to manage a set of sequential services to reduce the amount of monitoring data that will be processed by each AM to prevent bottlenecks. Therefore, constraint 3.3 ensures that parallel services are managed by different AMs. Constraint 3.4 guarantees that if an AM is assigned to at least one service, then it is considered as used. Constraints 3.5 and 3.6 are the binary restrictions of the decision variables.

$$min \sum_{m \in \mathcal{M}} y_m \tag{3.1}$$

Subject to:

$$\sum_{m \in \mathcal{M}} x_m^s = 1 \qquad \forall s \in \mathcal{S} \tag{3.2}$$

$$x_m^i + x_m^j <= 1 \qquad \forall m \in \mathcal{M},\ \forall (i,j) \in \mathcal{P} \tag{3.3}$$

$$x_m^s <= y_m \qquad \forall s \in \mathcal{S},\ \forall m \in \mathcal{M} \tag{3.4}$$

$$x_m^s \in \{0,1\} \qquad \forall s \in \mathcal{S},\ \forall m \in \mathcal{M} \tag{3.5}$$

$$y_m \in \{0,1\} \qquad \forall m \in \mathcal{M} \tag{3.6}$$

3.2.4 Need for Approximate Approach

To realize whether it is reasonable or not to solve the above-mentioned optimization model and come up with an optimal solution every time we have a new application to deploy in a public Cloud, let us consider how many deployments of applications in a public Cloud there are and how long it takes to resolve this optimization problem.

In a public Cloud such as OpenShift [76], there is a deployment of a new application every minute. In addition, according to new principles of application development, delivery, operation such as DevOps, and continuous delivery, for each application, there are multiple deployments per day. Furthermore, as we will show in the evaluation section (Section 3.4.3), the time needed to solve the optimization model using the CPLEX solver [77] is not acceptable. It can exceed two hours, and whenever the number of services goes beyond 250, the solver will not find the optimal solution. In fact, a large SBP may comprise hundreds or thousands of services [78, 79], so finding the optimal solution, in this context, is not possible. Consequently, to tackle this NP-hard problem [80], we introduce in the following section our first approach that aims to find a near-optimal number of AMs for the management of SBPs in a polynomial time.

3.3 AMs Optimization in SBP Graphs with Cycles

In this section, we present our first proposal for determining the appropriate number of AMs that will be used by SBP services for their management (Section 3.3.1). An illustrative example of our approach is then provided (Section 3.3.2). Afterwards, we present the experiments that we perform to evaluate it (Section 3.3.3).

3.3.1 Proposed Approach

The approach is based on three algorithms called LowerBound, ServiceRelatedParallelSets and AMsAssignement. Firstly, the LowerBound algorithm (*cf.* Algorithm 3.1) takes as inputs an SBP graph, and it aims at determining all the sets of services that can run in parallel. Secondly, the ServiceRelatedParallelSets algorithm (*cf.* Algorithm 3.2) takes as inputs the sets of parallel services, and it consists in determining, for each service, the set of services that can run in parallel with it. Thirdly, the AMsAssignement algorithm (*cf.* Algorithm 3.3) takes as inputs the set of services that can run in parallel with each service. It aims at assigning AMs to the SBP services while fulfilling the constraints given in equations 3.2 and 3.3.

To sum up, the proposed approach operates in three successive steps:

1. Determination of all sets of parallel services,
2. Construction of all sets of related services,
3. Assignment of AMs to the SBP services.

3.3.1.1 Determination of all sets of parallel services

Step 1 is implemented using the LowerBound algorithm. It starts by initializing the current set of parallel services with the initial services of the SBP graph and the lower bound number with the number of elements of the current set (*cf.* lines 1-2). The current set is added to the resulting sets of parallel services (*cf.* line 3). The algorithm iterates until determining all the sets of parallel services (*cf.* line 4 and line 15). To do so, the elements of the current set are assigned to the previous set, and the current set is initialized to the empty set (*cf.* lines 5-6). Next, it iterates over the set of immediate successors of the previous parallel services (*cf.* lines 7-8). For each node, whether its type is either an AND/OR -Join and all its predecessors have completed their execution or not an AND/OR -Join, LowerBound continues to explore the successors of the candidate node seeking a node of *service* type, and then the latter is added to the new set of services that can run in parallel (*cf.* lines 9-10). The algorithm checks whether the current set has not been determined in a previous iteration yet (*cf.* line 11). If it is the case, this set is added to the resulting sets of parallel services (*cf.* line 12), and the lower bound number of AMs is possibly updated (*cf.* lines 13-14). This number is equal to the maximum number of services that can run in parallel. LowerBound terminates when either the current set of services is empty or it is a subset of an existing set (*cf.* line 15).

Algorithm 3.1: LowerBound

Data: - $\mathcal{G}$ =(V,E,I,τ,ID): SBP graph
Result: - ParallelSets: Set of sets of parallel services
- lbn: Lower bound number of AMs

begin

1 $CurrentParallelSet \leftarrow \text{Services}(I)$;
2 $lbn \leftarrow |CurrentParallelSet|$;
3 $ParallelSets \leftarrow \{CurrentParallelSet\}$;
4 **repeat**
5 $PreviousParallelSet \leftarrow CurrentParallelSet$;
6 $CurrentParallelSet \leftarrow \emptyset$;
7 **for** $v_i \in PreviousParallelSet$ **do**
8 **for** $(v_i,v_j) \in E$ **do**
9 **if** ($\tau(v_j) \notin$ {AND-Join, OR-Join}) ***or*** ($\tau(v_j) \in$ {AND-Join, OR-Join} ***and*** $\text{AllPredecessors}(v_j)$) **then**
10 $CurrentParallelSet \leftarrow CurrentParallelSet \cup \text{Services}(v_j)$;
11 **if** $\nexists$ *set s.t.*($set \in ParallelSets$ ***and*** $CurrentParallelSet \subseteq set$) **then**
12 $ParallelSets \leftarrow ParallelSets \cup CurrentParallelSet$;
13 **if** $|CurrentParallelSet| > lbn$ **then**
14 $lbn \leftarrow |CurrentParallelSet|$;
15 **until** ($CurrentParallelSet = \emptyset$ ***or*** $\exists$ *set s.t.*($set \in ParallelSets$ ***and*** $CurrentParallelSet \subseteq set$));

3.3.1.2 Construction of all sets of related services

Step 2 is implemented using the ServiceRelatedParallelSets algorithm. It starts by determining the nodes of *service* type in the SBP graph (*cf.* line 2). After that, for each service (*cf.* lines 3-4), its corresponding set is initialized to the empty set (*cf.* line 5). The algorithm iterates over the sets of parallel services (*cf.* line 6), and if the current service belongs to the current set (*cf.* line 7), that is it can be run in parallel with the services belonging to the current set, then these services are added to its set of related services (*cf.* line 8).

Algorithm 3.2: ServiceRelatedParallelSets

Data: - $\mathcal{G}$ =(V,E,I,τ,ID): SBP graph
- ParallelSets: Set of sets of parallel services

Result: - ServiceRelatedParallelSets: Array of <serviceId,serviceSet>

begin

1 $i \leftarrow 1$;
2 $S \leftarrow$ ServiceNodes(V);
3 **for** $s \in S$ **do**
4 $ServiceRelatedParallelSets[i].serviceId \leftarrow s$;
5 $ServiceRelatedParallelSets[i].serviceSet \leftarrow \emptyset$;
6 **for** $set \in ParallelSets$ **do**
7 **if** $s \in set$ **then**
8 $ServiceRelatedParallelSets[i].serviceSet \leftarrow ServiceRelatedParallelSets[i].serviceSet \cup (set \setminus \{s\})$;
9 $i \leftarrow i + 1$;

3.3.1.3 Assignment of AMs to SBP services

Step 3 is implemented using the AMsAssignement algorithm. The algorithm starts by initializing the AM number that will be assigned to each service and the number of AMs to 0 (*cf.* lines 1-2). After that, the elements of *ServiceRelatedParallelSets* are sorted in decreasing order according to their cardinality to promote services with maximal number of services that can run in parallel with them (*cf.* line 3). Then for each service *s* (*cf.* line 5), the algorithm tries to assign an AM to it that fulfills the constraints given in equations 3.2 and 3.3 (*cf.* lines 6-12). To do so, it starts with the first AM (*cf.* line 6) and checks whether this AM might be assigned to *s* (*cf.* line 8); i.e., this AM has not been assigned to any service that can run in parallel with *s* yet. If it is the case, the current AM is assigned to *s* (*cf.* line 9). Otherwise, the algorithm just updates the current AM (*cf.* lines 10-11). This step is repeated until an AM is assigned to the current service (*cf.*

line 12). The resulting number of AMs is possibly updated (*cf.* lines 13-14).

Algorithm 3.3: AMsAssignement

Data: - $\mathcal{G}$ =(V,E,I,τ,ID): SBP graph
- ServiceRelatedParallelSets: Array of <serviceId,serviceSet>

Result: - AMs: Array containing the AM assigned to each service
- nbAMs: Number of AMs

begin

1 $AMs \leftarrow$ [vector of 0s];
2 $nbAMs \leftarrow 0$;
3 SortDecreasingOrderOfCardinalityOfSets($ServiceRelatedParallelSets$);
4 $S \leftarrow$ ServiceNodes(V);
5 **for** $i \in \{1, 2, \ldots |S|\}$ **do**
6 $currentAM \leftarrow 1$;
7 **repeat**
8 **if** $\nexists\ s$ *s.t.(*$s \in ServiceRelatedParallelSets[i].serviceSet$ ***and*** $AMs[s] = currentAM$*)* **then**
9 $AMs[ServiceRelatedParallelSets[i].serviceId] \leftarrow currentAM$;
10 **else**
11 $currentAM \leftarrow currentAM + 1$;
12 **until** $\nexists\ s$ *s.t.(*$s \in ServiceRelatedParallelSets[i].serviceSet$ ***and*** $AMs[s] = currentAM$*)*;
13 **if** $currentAM > nbAMs$ **then**
14 $nbAMs \leftarrow currentAM$;

According to the constraint given in equation 3.3, we assume that if the number of AMs provided by the AMsAssignement algorithm is equal to the lower bound number of AMs provided by the LowerBound algorithm, then the former number is optimal.

3.3.1.4 Theoretical complexity

Our proposed approach consists of three main steps: The first one consists in determining all the sets of parallel services (Algorithm 3.1). The worst case time complexity of this step is bounded by $O(n^4 \times 2^n)$, where n is the number of services that compose a given SBP. Thus, the number of iterations of the while loop is bounded by $O(2^n)$, and the time complexity of an iteration is bounded by $O(n^4)$. In fact, the number of sets of services that can run in parallel is not large. Therefore, we do not face the problem of determining all the possible combinations of services, where its time complexity is $O(n^4 \times 2^n)$.

The second step is to determine all the sets of related services (Algorithm 3.2). The worst case time complexity of this step is bounded by $O(n \times m)$, where n is the number of services, and m is the number of sets of parallel services.

The third step aims to assign AMs to the SBP services (Algorithm 3.3). The worst case time complexity of this step is bounded by $O(n \times log(n) + n^2) \simeq O(n^2)$, where n

is the number of services. We choose the QUICKSORT algorithm to sort *ServiceRelatedParallelSets*. This algorithm has a complexity of $O(n \times log(n))$.

Hence, the overall time complexity of the approach is: $O(n^4 \times 2^n)$.

3.3.2 Illustrative Example

Herein, we present an example illustrating how the proposed algorithms work. Let us consider the example depicted in Figure 3.2 that represents an SBP graph with a cycle.

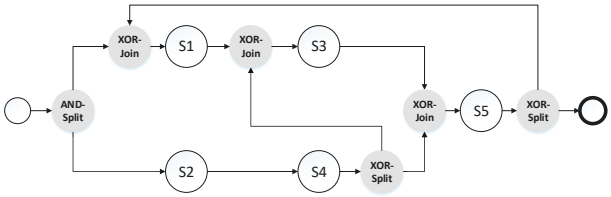


Figure 3.2: Example of SBP graph with cycle.

First, *CurrentParallelSet* is initialized to the set of the initial *services* of the graph, *i.e.* $\{S1, S2\}$, and lbn is set to *2*, which corresponds to the cardinality of *CurrentParallelSet*. Then $\{S1, S2\}$ is added to the resulting set *ParallelSets*. The repeat loop iterates until *CurrentParallelSet* is either empty or already added to *ParallelSets*.

The first iteration considers the set $\{S1, S2\}$. The successors of $S1$ and $S2$ of *service* type are added to the current set of parallel service, *i.e.* *CurrentParallelSet*=$\{S3, S4\}$. This set is added to *ParallelSets* because neither it is empty nor its elements belong to a set in *ParallelSets*. During iteration 2, iteration 3 and iteration 4, respectively, the set $\{S5, S3\}$, $\{S1, S5\}$ and $\{S3, S1\}$ is added to *ParallelSets*. The process goes on until reaching the 5^{th} and last iteration since the set $\{S5, S3\}$ has been already added to *ParallelSets* during iteration 2.

The LOWERBOUND algorithm comes to an end after five iterations and checking that all the sets of parallel services are determined. Thus, it yields the following output:

- *ParallelSets* $= \{\{S1, S2\}, \{S3, S4\}, \{S5, S3\}, \{S1, S5\}, \{S3, S1\}\}$
- $lbn = 2$

After that, SERVICERELATEDPARALLELSETS starts by determining the set of services that can run in parallel with $S1$. This set is the union of the sets of parallel services that contain $S1$, and it is equal to $\{S2, S5, S3\}$. In the second iteration, the set of services that can run in parallel with $S2$ is determined, which is $\{S1\}$. During iteration 3, iteration 4 and iteration 5, respectively, the set of services that can run in parallel with $S3$, $S4$ and $S5$ is determined: $\{S4, S5, S1\}$, $\{S3\}$ and $\{S3, S1\}$. The SERVICERELATEDPARALLELSETS algorithm yields the following output:

$< 1, \{S2, S5, S3\} >$	$< 2, \{S1\} >$	$< 3, \{S4, S5, S1\} >$	$< 4, \{S3\} >$	$< 5, \{S3, S1\} >$

Next, AMsAssignement sorts the elements of the above array in a decreasing order according to their cardinality: $< 1, \{S2, S5, S3\} >$, $< 3, \{S4, S5, S1\} >$, $< 5, \{S3, S1\} >$, $< 2, \{S1\} >$, $< 4, \{S3\} >$. In the first iteration, AM number 1, *i.e.* AM1, is assigned to $S1$. In the second iteration, since the service $S3$ can run in parallel with $S1$ as shown in the above table, AM1 can not be assigned to $S3$. In this case, AM2 is dedicated to $S3$. In the third iteration, the algorithm tries to assign AM1 and then AM2 to $S5$. These AMs can not be assigned to $S5$ because $S5$ can run in parallel with $S1$ and $S3$, and $AM1$ and $AM2$ have been already assigned to these services. Thus, AM3 is assigned to $S5$. AMsAssignement proceeds in the same manner to determine the AM that will be assigned to services $S2$ and $S4$. After 3 iterations, the algorithm comes to the end and returns the following results:

Services	$S1$	$S3$	$S5$	$S2$	$S4$
AM number	1	2	3	2	1

$nbAMs = 3$

3.3.3 Performance Evaluation

In this section, we present a series of performance evaluations of our proposed approach in terms of execution time and quality of the provided solutions based on the closeness between these solutions and the lower bound numbers of AMs outputted by the LowerBound algorithm. The first part is the experimental results conducted on two real datasets, and the second part is the performance results of experiments conducted on a dataset based on randomly generated graphs to push further experiments in order to generalize the performance results. All experiments are carried out on an Intel Core i5 PC with 2.53 GHz and 4GB of RAM. All the results are average values across 10 independent runs.

To the best of our knowledge, none of the existing works has explicitly focused on the determination of the optimal number of AMs for the management of SBPs in the Cloud (see Chapter 2 for more details). Therefore, we do not make comparisons with existing work here.

3.3.3.1 Experiments on public real datasets

We present in the following the performance results of experiments conducted on two public datasets of real business processes which are presented in an XML format. Each XML file stores the data of an SBP including services names and gateways (parallel, inclusive, and exclusive gateways), and the sequence flows between SBP elements.

- **Dataset 1** consists of 560 BPMN process models shared by the IBM Business Integration Technologies (BIT) team [81]. At most, there are 195 services, 139

gateways, and 326 edges in a process (Table 3.1).

	Min.	Max.
Number of start events	1	32
Number of end events	1	32
Number of services	1	195
Number of gateways	1	139
Number of edges	2	326

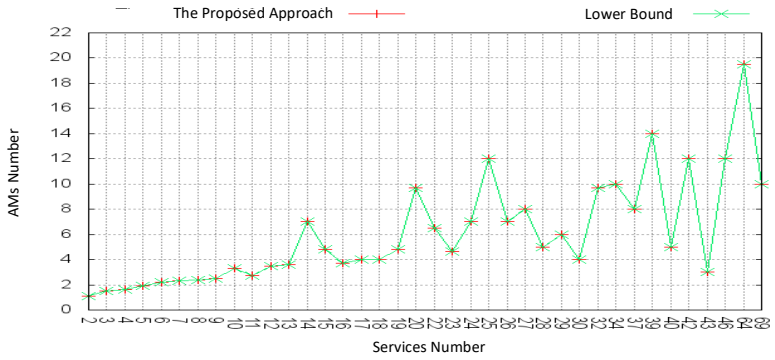
Table 3.1: Details of the IBM dataset.

- **Dataset 2** consists of 205 EPC process models from the SAP reference models represented in an EPC Markup Language (EPML) format [82]. At most, there are 43 services, 39 gateways, and 125 edges in a process (Table 3.2).

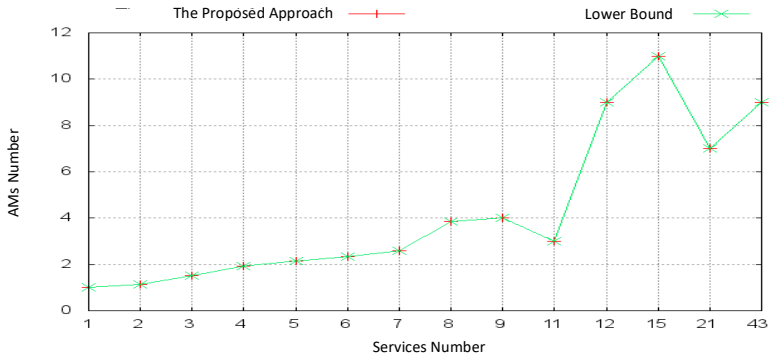
	Min.	Max.
Number of events (start, end, intermediate)	2	67
Number of services	1	43
Number of gateways	0	39
Number of edges	2	125

Table 3.2: Details of the SAP dataset.

At this level, we recall our assumption that considers that if the number of AMs provided by the AMsAssignement algorithm is equal to the lower bound number of AMs provided by the LowerBound algorithm, the latter number is optimal. According to this assumption, our approach provides optimal solutions for all the considered real SBPs (Figure 3.3) in a reasonable time that does not exceed 0.1 second (Figure 3.10a).



(a) IBM dataset



(b) SAP dataset

Figure 3.3: First proposal: Number of AMs versus number of services - Experiments on IBM and SAP datasets.

3.3.3.2 Experiments on randomly generated dataset

SBPs are generally modeled as sparse graphs where nodes indicate services to invoke and gateways and directed edges represent dependencies between nodes. In order to consider a realistic dataset, we generate different connected SBP graphs with 10-100 services, in increments of 10, and a number of edges ranging from $n-1$ to $3.2 \times (n-1)$ (*i.e.* 320% of $(n-1)$), in increments of 0.2, where n is the order of the graph. Indeed, we do not generate graphs with more edges since whenever the number of edges goes beyond $2.4 \times (n-1)$, the lower bound number of AMs is equal to n. Thus, the number of AMs that will be used by n SBP services is equal to n. Therefore, in our experiment, for each graph of order n, we consider 12 densities (from 100% of $(n-1)$ to 320% of $(n-1)$).

As shown in Figure 3.4, when the number of edges of an SBP graph of order n goes beyond $2.4 * (n-1)$, the lower bound number of AMs is equal to n. For this reason, we limit our evaluation of the quality of the results provided by the AMsAssignement algorithm to graphs with $n-1$ – $2.4 \times (n-1)$ edges.

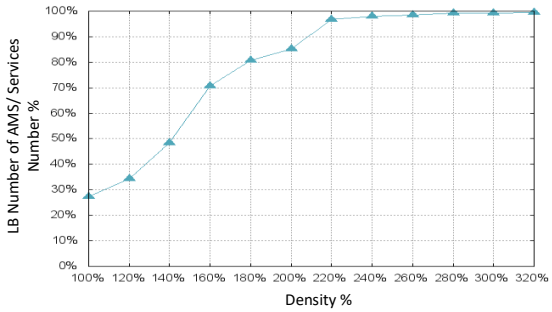


Figure 3.4: First proposal: Lower bound number of AMs / services number (in %) versus SBP graph density - Experiments on randomly generated dataset.

Although the time complexity of the proposed approach is exponential (in fact, it is equal to $O(|ParallelSets|)$ that is bounded by $O(2^n)$), where n is the number of SBP services, the approach is useful in practice because the number of sets of services that can run in parallel ($|ParallelSets|$) is usually far less than the number of sets under consideration. The results depicted in Figure 3.5 show that the experimental variations of $|ParallelSets|$ versus the number of services are linear with a low slope. For example, for SBP graphs with 50 services, the number of sets of parallel services does not exceed 23. We note that the execution time of the approach is very small and does not exceed 0.24 second.

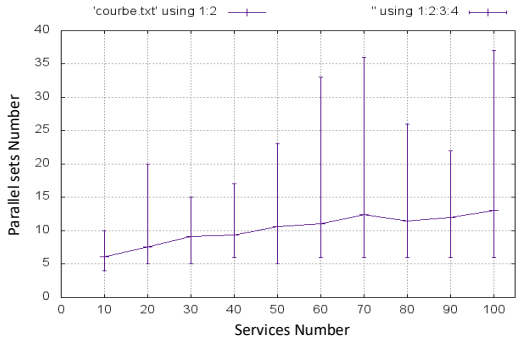


Figure 3.5: First proposal: Cardinality of ParallelSets versus number of services - Experiments on randomly generated dataset.

Based on our assumption, our algorithm outputs optimal solutions for 94% instances of the randomly generated graphs and near-optimal solutions for the remaining instances (6% of the considered graphs). As depicted in Figure 3.6, the difference between the optimal solution and the solution provided by our algorithm does not exceed 6 AMs, where for 49% of them, the difference is equal to one AM. It is interesting to note, as it is shown in Figure 3.2, that the optimal number of AMs for the management of an SBP may be greater than the lower bound number of AMs, which means that our approach

solves more than 94% of the considered graphs to optimality.

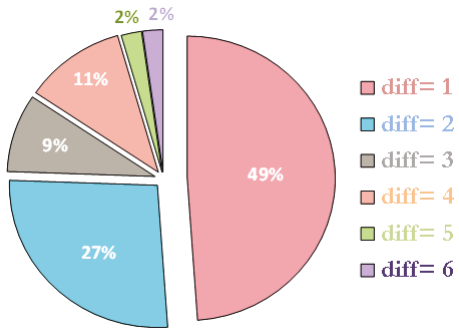


Figure 3.6: First proposal: Difference between number of AMs provided by AMsAssignement algorithm and lower bound number of AMs (in %) - Experiments on randomly generated dataset.

3.4 AMs Optimization in SBP Graphs Without Cycles

In this section, we present our second attempt that aims to determine the appropriate number of AMs for the management of large SBPs. In this proposal we focus on large SBPs that can be modeled as Directed Acyclic Graphs (DAGs), since a large percentage of SBPs are usually expressed as DAGs. Our first proposal introduced in the above section deals with the different types of gateways (*e.g.* AND, OR, XOR) in the same way; i.e., the outgoing services of each gateway are always considered as parallel services even for alternative services, which are the outgoing services of a *XOR-split* gateway. This proposal is not effective when the given SBP graph contains *XOR-split* gateways. Because of this, we suggest a second proposal that attempts to further reduce the number of AMs by focusing on *XOR-split* gateways. We first introduce our proposed approach (Section 3.4.1). We then present an illustrative example of how it works (Section 3.4.2). Thereafter, we introduce the experiments that we perform to evaluate our second proposal (Section 3.4.3).

3.4.1 Proposed Approach

Our second approach is based on two algorithms called Compaction and AMsAssignement. The Compaction algorithm, whose pseudo-code is given in Algorithm 3.4 takes as input an SBP graph and outputs a compact graph. It mainly aims to (i) merge the outgoing paths of each *XOR-Split* gateway into a single path in order to assign only one AM to alternative services, and (ii) remove all gateways, regardless of their types, to avoid assigning AMs to them. The AMsAssignement algorithm, whose pseudo-code is given in Algorithm 3.5, has as an input a compact graph and outputs the AM assigned

to each service. It aims to assign AMs to the SBP services while fulfilling the constraints given in equations 3.2 and 3.3.

3.4.1.1 Compaction Algorithm

The COMPACTION algorithm operates in two successive steps: Step 1 is to determine the set of nodes of the compact graph (*cf.* lines 1-31). Step 2 is to discover the dependency relationships between these nodes (*cf.* lines 32-39).

It starts by initializing the nodes, the initial nodes, and the edges sets of the output graph G_1 denoted, respectively, by V_1, I_1, and E_1 to the empty set and the current set *CurrSet* to the set of initial nodes of the input graph G (*cf.* lines 1-2). *CurrSet* is a set of meta-nodes where each meta-node consists of a set of alternative services; i.e., only one of these services will be executed all along the SBP life cycle. In the first step, COMPACTION iterates until all the nodes of G_1 are determined (*cf.* line 3). To do so, it checks whether all the candidate meta-nodes that belong to *CurrSet* consist of only nodes of *service* type in order to remove all gateways, regardless of their types (*cf.* lines 4-6). If it is the case, these meta-nodes are added to V_1 (*cf.* lines 28-30), and the successors of each one are merged into a single meta-node which is added to the new *CurrSet* (*cf.* line 31). Otherwise, there is at least one candidate meta-node that contains gateways (*cf.* lines 6-27). In this case, the algorithm goes through the set of nodes that composes the candidate meta-node in order to remove gateways from it (*cf.* line 7). If the current node is an *AND/OR -Split* (*cf.* lines 8-17), in fulfillment of the constraint given in equation 3.3, different AMs should be assigned to the successors of this node. To do so, these nodes are added to different meta-nodes in *CurrSet* (*cf.* lines 11-15), and, if necessary, a new meta-node is added to *CurrSet* to make sure that parallel services belong to different meta-nodes (*cf.* lines 16-17). Otherwise, the type of the gateway is not an *And/Or -Split* (*cf.* lines 18-27), so only one of its successors will be executed, and then one AM is sufficient to manage them. The algorithm checks whether all its predecessors have completed their execution (*cf.* line 21). If it is the case, it merges all the successors of the current gateway into the current meta-node (*cf.* line 22). Otherwise, since the gateway will be removed (*cf.* line 20), it looks to merge the current meta-node with another one (*cf.* lines 24-27).

After determining the set of meta-nodes V_1 of the compact graph, COMPACTION looks for determining the dependency relationships between these meta-nodes (*cf.* lines 32-39). For each meta-node u (*cf.* line 32), it checks whether all the nodes constituting this meta-node belong to the set of initial nodes of G. If so, u is added to the set of initial nodes I_1 (*cf.* lines 34-35). Otherwise, the algorithm determines the predecessors of the nodes constituting u (*cf.* lines 36-38) and the edge connecting u, and each meta-node consisting of at least one of these predecessors is added to E_1 (*cf.* line 39).

Algorithm 3.4: COMPACTION

```
Data:   - G =(V,E,I,τ,ID): SBP graph
Result: - G1 =(V1,E1,I1,τ1,ID): SBP graph
begin
                    - - - - - - - - - - - - - - - - Step 1 - - - - - - - - - - - - - - - -
1     V1, I1, E1 ← ∅;
2     CurrSet ← I;
3     while CurrSet ≠ ∅ do
4         setOfMetaServices ← false;
5         while not(setOfMetaServices) do
6             if ∃ MetaNode s.t.(MetaNode ∈
                CurrSet and not(IsSetOfServices(MetaNode))) then
7                 for v ∈ MetaNode do
8                     if τ(v) ∈ {AND-Split,OR-Split} then
9                         MetaNode ← MetaNode \ {v};
10                        for u ∈ Successor({v}) do
11                            isAdded ← false;
12                            for Node ∈ CurrSet where not(isAdded) do
13                                if ParentNodeIsXor({u},Node) then
14                                    Node ← Node ∪ {u};
15                                    isAdded ← true;
16                            if not(isAdded) then
17                                CurrSet ← CurrSet ∪ {u};
18                    else
19                        if τ(v) ≠ service then
20                            MetaNode ← MetaNode \ {v};
21                            if Predecessor({v}) ⊆ V1 then
22                                MetaNode ← MetaNode ∪ Successor({v});
23                            else
24                                for Node ∈ CurrSet do
25                                    if ParentNodeIsXor(MetaNode,Node) then
26                                        MetaNode ← MetaNode ∪ Node;
27                                        CurrSet ← CurrSet \ Node;
28            else
29                setOfMetaServices ← true;
30        V1 ← V1 ∪ CurrSet;
31        CurrSet ← Successor(CurrSet);
                    - - - - - - - - - - - - - - - - Step 2 - - - - - - - - - - - - - - - -
32    for u ∈ V1 do
33        τ1(u) ← service;
34        if InitialMeta-node(u) then
35            I1 ← I1 ∪ {u};
36        else
37            for v ∈ V1 do
38                if ∃ s s.t.(s ∈ Predecessor(u) and s ∈ v) and (v,u) ∉ E1 then
39                    E1 ← E1 ∪ {(v,u)};
```

3.4.1.2 AMsAssignement Algorithm

At first, the AMsAssignement algorithm initializes the AM number that will be assigned to each service and the number of AMs to 0 (*cf.* lines 1-2). After that, it tries to assign AMs to the SBP services that fulfill the constraints given in equations 3.2 and 3.3 (*cf.* lines 3-20). It starts with the initial nodes of the compact graph where different AMs are assigned to these nodes (*cf.* lines 3-5). Then the current set of parallel services is initialized to the graph's initial nodes (*cf.* line 6). The algorithm iterates until all the services have AMs assigned to them (*cf.* line 7). To do so, it looks for the successors of the current parallel services, and for each service, it checks whether all its predecessors have completed their execution (*cf.* lines 8-11). If it is the case, this service is added to the set of next parallel services (*cf.* line 12), and if the AM assigned to its predecessor service is not used by another services that can run in parallel with it (*cf.* line 13), then this AM is assigned to it in order to assign the same AM to sequential services (*cf.* line 14). Otherwise, the algorithm updates the current AM with an AM that is not already assigned to any service that can run in parallel with the current service. This AM is then assigned to it (*cf.* lines 15-17). The total number of AMs needed to manage the SBP is possibly updated (*cf.* lines 18-19).

3.4.1.3 Theoretical complexity

Our proposed approach consists of two algorithms. The first one aims to compact an SBP graph by merging the outgoing paths of each *XOR-Split* gateway into a single path and removing all gateways, regardless of their types (Algorithm 3.4). It consists of two main steps. The first step is to determine the set of nodes of the compact graph. The worst case time complexity of this step is bounded by $O(m^6)$, where m is the number of nodes in the graph. The second step is to determine the dependency relationships between the resulting nodes. The worst case time complexity of this step is bounded by $O(n^3)$, where n is the number of services. The second algorithm aims to assign AMs to the SBP services (Algorithm 3.5). Its worst case time complexity is bounded by $O(n^4)$, where n is the number of services.

Therefore, the overall time complexity of the approach is: $O(m^6+n^3+n^4) \simeq O(m^6)$.

3.4.2 Illustrative Example

Herein, we present an example illustrating how the proposed algorithms work. Let us consider the example depicted in Figure 3.7 that represents an SBP composed of ten services (see Figure 3.7 (left)).

Based on the characteristics of SBPs, we assume that an SBP is split into levels that can be executed sequentially, and services belonging to a level are independent from each other. The compact graph is in fact an intermediary graph, which is used only to determine the appropriate number of AMs. It consists of a set of meta-nodes where a meta-node is a group of alternative services. The relationship between levels of services

Algorithm 3.5: AMsAssignement

Data: - $\mathcal{G}$ = (V,E,I,τ,ID): SBP graph
Result: - AMs: Array containing the AM assigned to each service
- nbAMs: Number of AMs

begin

1 $AMs \leftarrow$ [vector of 0s];
2 $nbAMs \leftarrow 0$;
3 **for** $s \in I$ **do**
4 $nbAMs \leftarrow nbAMs + 1$;
5 AMs[ID of s] $\leftarrow nbAMs$;
6 $CurrentParallelSet \leftarrow I$;
7 **while** $CurrentParallelSet \neq \emptyset$ **do**
8 $NextParallelSet \leftarrow \emptyset$;
9 **for** $s_i \in CurrentParallelSet$ **do**
10 **for** $(s_i,s_j) \in E$ **do**
11 **if** $\nexists$ s *s.t.*$(s \in$ Predecessor$(\{s_j\})$ ***and*** AMs*[ID of s]*$=0)$ **then**
12 $NextParallelSet \leftarrow NextParallelSet \cup \{s_j\}$;
13 **if** $\nexists$ s *s.t.*$(s \in$ Successor$(\{s_i\})$ ***and*** AMs*[ID of s]* $= AMs$*[ID of* s_i*]*$)$ **then**
14 AMs[ID of s_j] $\leftarrow AMs$[ID of s_i];
15 **else**
16 $currentAM \leftarrow$ GetAM(s_j);
17 AMs[ID of s_j] $\leftarrow currentAM$;
18 **if** $nbAMs < currentAM$ **then**
19 $nbAMs \leftarrow currentAM$;
20 $CurrentParallelSet \leftarrow NextParallelSet$;

is kept in this graph. Level 1 contains the initial nodes of the input graph *G*. Level $L+1$ contains the successors of the services of level L. Hence, the dependencies in this graph do not reflect the execution dependencies between individual services, but rather the dependencies between levels.

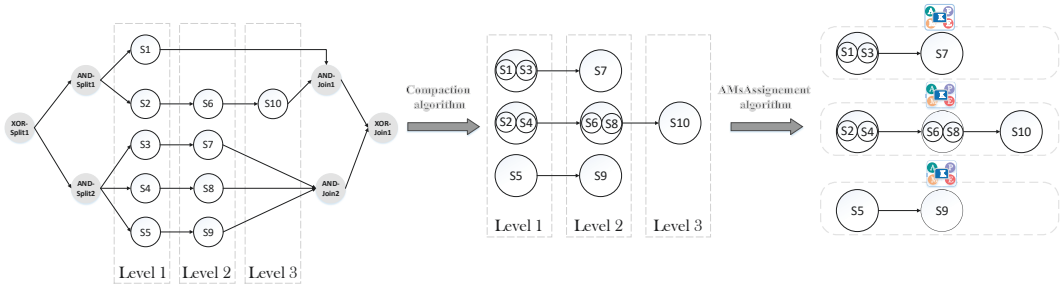


Figure 3.7: Example of SBP graph without cycles where XOR-Split preceding two AND-Splits (left) and result of applying Algorithms 3.4 and 3.5 on it.

In the first step, COMPACTION initializes *CurrSet* to the set of the initial nodes of the input graph *CurrSet*={{XOR-split1}}. The while loop iterates over *CurrSet* until the latter is empty. The first iteration considers the meta-node {*XOR-split1*}. This meta-node does not consist only of nodes of *service* type. In this case, the algorithm goes through these nodes; and for each one, in case it is not a *service*, this node is replaced by its successors if and only if, the predecessors of these nodes have been processed earlier. *XOR-split1* is replaced by its successors, *i.e.* *CurrSet*={{AND-split1, AND-split2}}.

The second iteration considers the meta-node {{*AND-split1*, *AND-split2*}}. The node *AND-split1* is replaced by its successors *S1* and *S2* because *S1* and *S2* can run in parallel. These nodes are added to two different meta-nodes, *i.e.* *CurrSet*={{S1, AND-split2},{S2}}.

The third iteration considers the meta-node {*S1*, *AND-split2*}. The node *AND-split2* is replaced by its successors: *S3*, *S4*, and *S5*, which are added to different meta-nodes, *i.e.* *CurrSet*={{S1,S3},{S2,S4},{S5}}. All the meta-nodes in *CurrSet* consist of only nodes of *service* type. These meta-nodes are added to the set of nodes of the compact graph, *i.e.* V_1={{S1,S3},{S2,S4},{S5}}; and the successors of each meta-node are merged into a meta-node that is added to the new *CurrSet*: *CurrSet*={{AND-join1,S7},{S6,S8},{S9}}.

The fourth iteration considers the meta-node {*AND-join1*, *S7*}. The node *AND-join1* is removed form this meta-node because its predecessor *S10* has not been processed yet, *CurrSet*={{S7},{S6,S8},{S9}}. All the meta-nodes in *CurrSet* consist of only nodes of *service* type. These meta-nodes are added to V_1 and the successors of each one are merged into a meta-node that is added to the new *CurrSet*: *CurrSet*={{AND-join2},{S10,AND-join2}}.

The same process is applied to the current set {{*AND-join2*},{*S10,AND-join2*}}. The meta-node *{S10}* is added to V_1, and the algorithm comes to an end as $CurrSet = \emptyset$

To conclude the first step, the resulting set of the nodes of the compact graph is V_1={{$S1$,$S3$},{$S2$,$S4$},{$S5$},{$S7$},{$S6$,$S8$},{$S9$},{$S10$}}.

In the second step, COMPACTION iterates over the set of resulting meta-nodes V_1 to determine the dependency relationships between them. The first iteration considers the meta-node {*S1*,*S3*}. Since *S1* and *S3* are initial nodes in the input graph (Figure 3.7 (left)), this meta-node is added to the set of initial nodes of the compact graph I_1. The same process is applied to the meta-nodes {*S2*,*S4*} and {*S5*} that are then added to I_1.

The next iteration considers the meta-node {*S7*}. *S3* is a predecessor of it, then an edge connecting the meta-node {*S7*} and the meta-node that contains *S3* is added to the set of the edges of the compact graph E_1, *i.e.* E_1={({*S1*,*S3*},{*S7*})}. Similarly, the rest of the edges are determined. Therefore, we get the following set of results:

E_1={({$S1$,$S3$},{$S7$}),({$S2$,$S4$},{$S6$,$S8$}),({$S5$},{$S9$}), ({$S6$,$S8$},{$S10$})}.

I_1={{$S1$,$S3$},{$S2$,$S4$},{$S5$}}.

Applying the AMSASSIGNEMENT algorithm on the compact graph, three AMs are needed for the management of the given SBP. The first AM is assigned to the services $S1$, $S3$, and $S7$. The second AM is assigned to the services $S2$, $S4$, $S6$, $S8$, and $S10$, while the third one is assigned to the services $S5$ and $S9$.

Applying our approach on the SBP graph example depicted in Figure 3.8 (left), two AMs are enough for managing SBP. The first AM is dedicated to managing the services $S1$, $S2$, $S6$, and $S10$, while the second AM is assigned to the services $S3$, $S4$, $S5$, $S7$, $S8$, and $S9$.

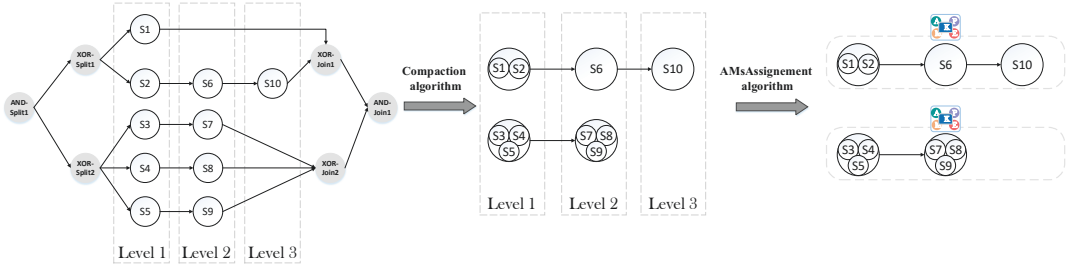


Figure 3.8: Example of SBP graph without cycles where AND-Split preceding two XOR-Splits (left) and result of applying Algorithms 3.4 and 3.5 on it.

3.4.3 Performance Evaluation

This section presents a series of performance evaluations of our second proposed approach in terms of execution time and quality of the provided solutions. In order to show the efficiency of this approach, we compare it to our first approach presented in Section 3.3 to verify how it enhances the AMs optimization process for the management of large SBPs. We also compare the solution provided by this approach to the solution obtained by solving the optimization model presented in Section 3.2.3 using the CPLEX solver [77]

when such a solution is possible. Let us denote by:

- S_a the solution provided by our second proposed approach;
- S^* the optimal solution provided by CPLEX;
- G the gap between S_a and S^* (in %). Note that if $G = 0\%$, it means that S_a is optimal. G is defined as follows:

$$G = \frac{S_a - S^*}{S^*} \times 100$$

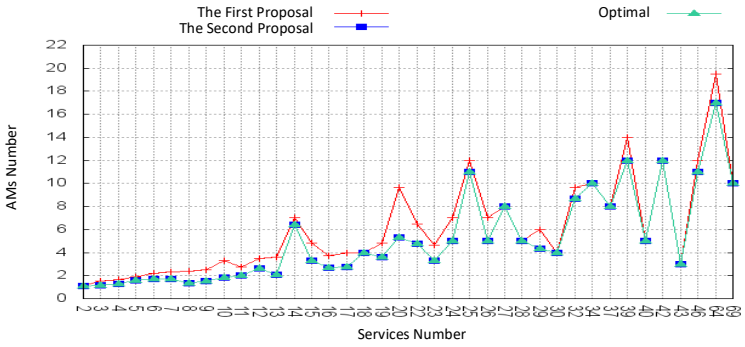
The first part of the series of performance evaluations is the experimental results conducted on two real datasets (Section 3.4.3.1), and the second part is the performance results of experiments conducted on a realistic dataset based on randomly generated graphs (Section 3.4.3.2). It is worth mentioning that, to the extent of our knowledge, there is no available public dataset of large business process models. Thus, to perform experiments on large SBPs, we generate a dataset of random large SBPs. All the results are average values across 10 independent runs.

The different experiments are carried out on an Intel Core $i7$ PC with 2.70 GHz and 8GB of RAM. We use the commercial CPLEX solver 12.6 to solve the ILP formulation (Section 3.2.3).

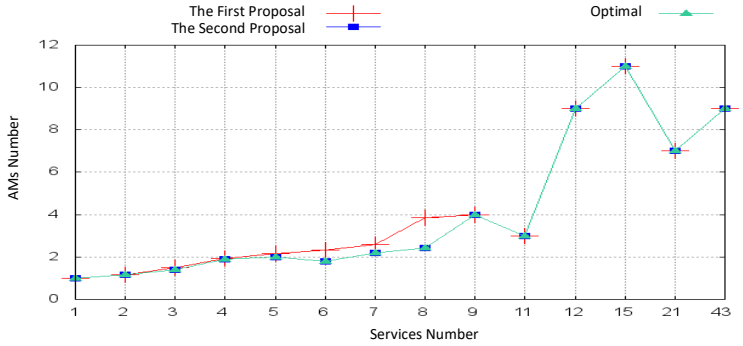
3.4.3.1 Experiments on public real datasets

Herein, we present the performance results of experiments conducted on IBM and SAP datasets (see Section 3.3.3.1 for more datasets details). In this proposal we focus on SBPs that can be modeled as DAGs. In fact, 85% of the IBM dataset and 94% of the SAP dataset are SBPs that can be represented as DAGs.

Experiment results on real datasets, depicted in Figure 3.9 show, our second proposal compared to the first one. It can be seen that the second proposal outperforms our first one in terms of reducing the number of AMs for the management of SBPs. It reduces this number by 18% and 7% for for IBM and SAP datasets, respectively. On the other hand, it can always find the optimal solution for the SBP models in IBM and SAP datasets in a short time, which does not generally exceed 0.06 second, as depicted in Figure 3.10.

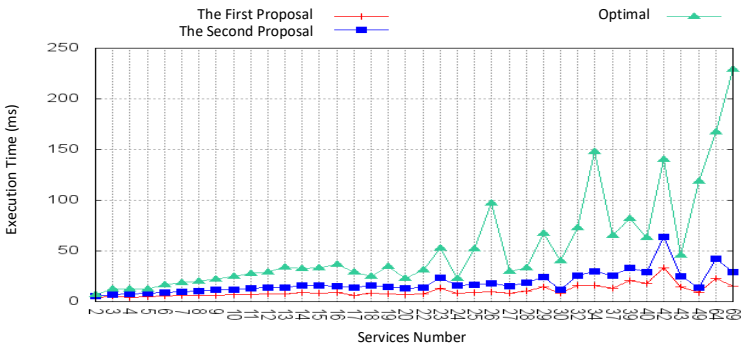


(a) IBM dataset

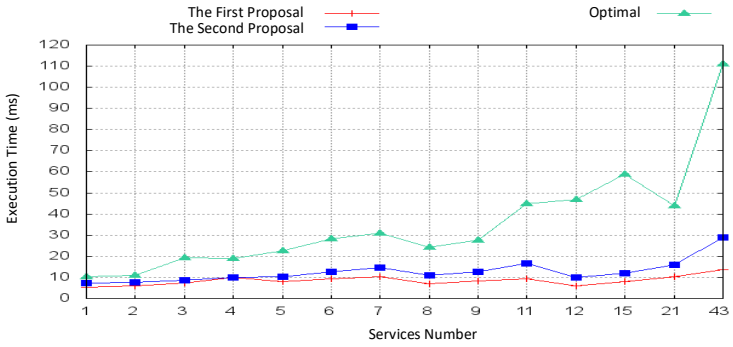


(b) SAP dataset

Figure 3.9: Second proposal: Number of AMs versus number of services - Experiments on IBM and SAP datasets.



(a) IBM dataset



(b) SAP dataset

Figure 3.10: Second proposal: Execution time versus number of services - Experiments on IBM and SAP datasets.

The results on real datasets show that our second proposal has good performance in terms of both quality and execution time. It provides the optimal solution for all the considered SBPs in few milliseconds. In the following, we perform experiments on randomly generated small and large graphs. On the one hand, we highlight that our generated dataset behaves the same way as real datasets. On the other hand, we evaluate our approach on large SBPs, which is our main goal here.

3.4.3.2 Experiments on randomly generated dataset

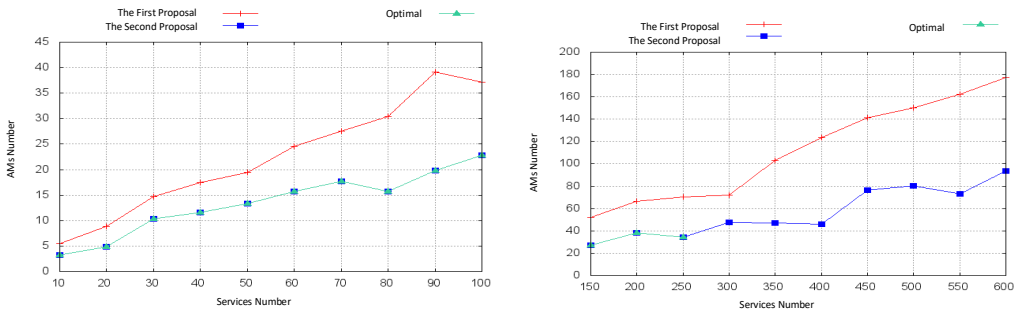
In order to evaluate the performance of the proposed approach on large SBPs, we implement a generator of random structured SBPs. The generation of these SBP graphs is based on Definition 3.2.2. Note that structured processes represent a class of business processes widely used in industry and academia [73]. We generate SBP graphs with two different sizes (small, large), with a number of services varying, respectively, from 10 to 100, in increments of 10, and from 150 to 600, in increments of 50. These graphs are sparse because SBP graphs are generally sparse (see IBM and SAP dataset details in Tables 3.1 and 3.2). The characteristics of the considered SBP graphs are reported in Table 3.3, where the nodes include both services and gateways (AND-split, AND-join, OR-split, OR-join, XOR-split, and XOR-join).

Experiment results on the randomly generated dataset are illustrated in Figure 3.11. Compared to the first approach, it can be seen that our second proposal outperforms the first one in terms of reducing the number of AMs. It decreases this number by 41%. On the other hand, it is able to find the optimal solution for 99% of the considered SBPs. We note that the gap between our approximate solution provided by the second proposal and the optimal solution is very small and does not exceed 0.75%. It means that our solution is very close to the optimal solution. As for the execution time of our second proposal, the results presented in Figure 3.12 show that its execution time remains reasonable even for

Graph Sizes	Services	Nodes	Edges
Small	10	16 - 22	19 - 30
	20	32 - 46	42 - 59
	30	48 - 72	60 - 100
	40	70 - 100	85 - 122
	50	90 - 124	112 - 152
	60	114 - 134	130 - 171
	70	126 - 150	155 - 214
	80	150 - 182	184 - 220
	90	164 - 198	203 - 260
	100	174 - 210	227 - 275
Large	150	286 - 348	369 - 432
	200	356 - 424	458 - 544
	250	474 - 524	604 - 673
	300	582 - 636	754 - 814
	350	658 - 748	850 - 950
	400	760 - 846	967 - 1084
	450	854 - 966	1099 - 1228
	500	972 - 1036	1243 - 1316
	550	1078 - 1152	1384 - 1486
	600	1178 - 1258	1494 - 1602

Table 3.3: Characteristics of generated small and large SBP graphs.

large SBPs. In fact, it does not exceed 12 seconds, whereas the CPLEX solver takes more than 2 hours to solve the problem; and whenever the number of services goes beyond 250, the solver can not find the optimal solution due to a problem of memory.



(a) Small graphs

(b) Large graphs

Figure 3.11: Second proposal: Number of AMs versus number of services - Experiments on randomly generated dataset.

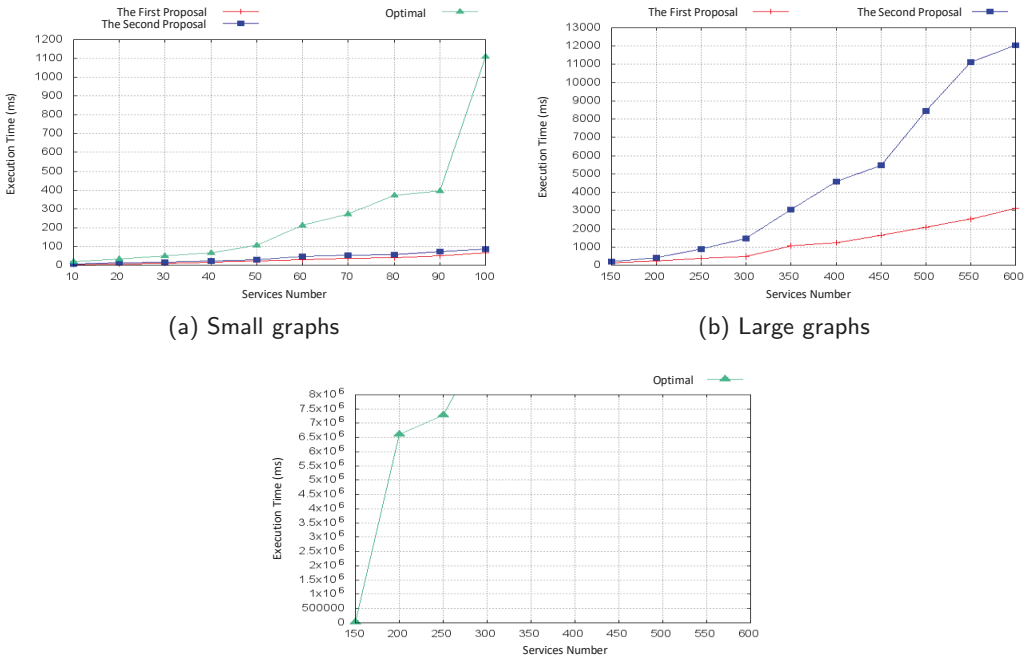


Figure 3.12: Second proposal: Execution time versus number of services - Experiments on randomly generated dataset.

Experiment results on the randomly generated dataset show that our second proposal has good performance in terms of both quality and execution time. It returns an optimal solution for more than 99% of the considered small and large SBPs in a reasonable time. Although these series of experiments are performed on a randomly generated dataset, we advocate that this dataset is realistic. For small SBP graphs, it behaves the same way as IBM and SAP real datasets, as shown in Figures 3.10 and 3.12a. Therefore, based on the generated dataset, we can conclude that the computational time of CPLEX increases quickly with the number of SBP services (Figure 3.12c). In fact, considering the example of the public Cloud OpenShift where there are few deployments of a new application every minute, we cannot use the CPLEX solver to determine the optimal number of AMs because it can take more than two hours to solve the model. However, we can apply our approaches that come up with near-optimal solutions in an acceptable time, which is of the order of a few milliseconds for small SBP graphs, and it does not exceed 12 seconds for large graphs.

To sum up, our two proposals are complementary; and based on them, we cover all types of SBPs, i.e. small and large SBPs that are represented as graphs with or without cycles. The second proposal is better than the first one for both small and large

SBPs expressed as DAGs. Thus, if the given SBP graph contains cycles, we use our first proposal for determining the appropriate number of AMs that will be used by services for their management; otherwise, the second proposal will be the most appropriate and effective method to solve the problem.

3.5 Conclusion

In this chapter, we focused on the optimization of the number of AMs in SBPs while avoiding management bottlenecks. We formally represented SBPs as graphs that allow us to preserve the semantics of business processes. Then we proposed an ILP formulation to solve this problem. Since solving the ILP formulation is time-consuming for large SBPs, we also proposed two different approaches for SBPs that are represented as graphs, respectively with and without cycles. Their main idea is to manage parallel services with different AMs in order to reduce the number of allocated AMs and avoid management bottlenecks. An illustrative example of how each approach works was given. Experiment results showed that our proposals obtain very encouraging results.

After facing the challenges of optimizing the number of AMs for the management of SBPs, since an AM consists of four basic components (monitor, analyzer, planner, and executor), we were motivated to extend our work to break down the AM into its main components. We are interested in optimizing the number of monitors, analyzers, planners, and executors separately. Hence, in the next chapter, we will focus on how to formulate this problem and solve it in a polynomial time.

Chapter 4

Autonomic Managers Components Optimization in Timed SBPs

Contents

4.1 Introduction . . . 51
4.2 Preliminaries: Monitor, Analyzer, Planner, Executor . . . 52
4.3 Problem Description and Formulation . . . 53
4.3.1 Problem Description . . . 53
4.3.2 Timed SBP Modeling . . . 54
4.3.3 Notations and Assumptions . . . 55
4.3.4 Problem Formulation . . . 56
4.4 Proposed Approach . . . 58
4.4.1 Approach Overview . . . 58
4.4.2 ParallelServices Algorithm . . . 58
4.4.3 M-A-P-E_Assignment Algorithm . . . 60
4.4.4 Theoretical Complexity . . . 66
4.5 Illustrative Example . . . 67
4.6 Performance Evaluation . . . 71
4.7 Conclusion . . . 75

4.1 Introduction

Executing SBPs in the Cloud requires autonomic management to cope with the dynamism and scalability of Cloud environments. In the previous chapter, we proposed an approach that aims to determine the appropriate number of AMs for the management of SBPs

such that this number is minimized while avoiding management bottlenecks. Since an AM consists of four basic components: a monitor, an analyzer, a planner, and an executor, which are not used with the same frequency. Thus, to go beyond the boundaries of an AM that is seen as a single resource, we propose in this chapter a new approach that consists in determining the appropriate number of monitors, analyzers, planners, and executors for the management of timed SBPs to further reduce the management costs.

This chapter is organized as follows: Preliminaries on the four basic components of an AM are firstly introduced in Section 4.2. Then we describe the problem that we tackle in this chapter, and we suggest an ILP formulation to solve it in Section 4.3. Our proposed approach as well as an illustrative example of how it works are presented, respectively, in Sections 4.4 and 4.5. Finally, the experimental results are detailed in Section 4.6.

4.2 Preliminaries: Monitor, Analyzer, Planner, Executor

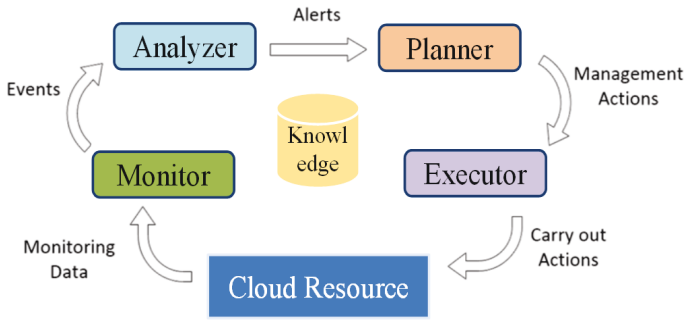


Figure 4.1: Autonomic control loop for Cloud resource.

Autonomic management resource for Cloud resources is depicted in Figure 4.1. In this autonomic loop, the managed element represents any hardware or software Cloud resource for which we want to endow an autonomous behavior. A shared knowledge is essential to maintain data of the managed resource, adaptation goals, and other information needed by the AM components. The different components of an AM are defined as:

1. **The Monitor** is used to periodically gather monitoring data from the managed resource;

2. **The Analyzer** is in charge of periodically analyzing monitoring data and checking whether an adaptation is required. If so, it sends an alert (as an event) to the planner;

3. **The Planner** is responsible for producing adaptation plans. An adaptation plan represents a workflow of elementary adaptation actions described in the knowledge base, which are needed to achieve the resource goals;

4. **The Executor** is responsible for carrying out the adaptation actions provided by the planner over the managed resource.

4.3 Problem Description and Formulation

In this section, we first show how to map a timed SBP to a directed graph. After that, we define the research problem that we tackle in this chapter. We then introduce the notations that are used throughout the problem formulation as well as the assumptions. Finally, we give an ILP formulation to solve the problem.

4.3.1 Problem Description

Managing SBPs in the Cloud involves using AMs to cope with the dynamism of Cloud environments. In fact, an AM consists of four basic components which are the monitor, the analyzer, the planner, and the executor that are not used with the same frequency. For example, the monitor is used much more frequently than the executor. Then we can imagine that assigning the same monitor to services that can run in parallel may lead to management bottlenecks, but this is not necessarily the case for the executor. Thus, we can share a component between different parallel services in order to further reduce the management costs while ensuring that each component is assigned to only one SBP service at a time. Consequently, to go beyond the boundaries of an AM that is seen as a single resource, it is interesting to minimize the number of monitors, analyzers, planners and executors separately while avoiding management bottlenecks.

In fact, the authors in [75, 16] adopted the principle of using multiple AMs for the management of applications to avoid management bottlenecks, but they did not provide any means to optimize their number. In our work, an AM component is able to manage a set of services, while it is assigned to only one service at a time to reduce the amount of monitoring data that will be processed by each AM to prevent bottlenecks. To this end, it is essential to determine all sets of services that can run in parallel.

In order to determine the optimal number of monitors that will be used by the SBP services for their management, we are required to know the monitoring time and frequency for each service to figure out the set of time stamps at which we can monitor each service in an attempt to share a monitor between different parallel services while ensuring that this monitor is used by only one service at a time. While the determination of the optimal number of analyzers requires us to know the set of time stamps at which we can analyze each service by means of the analysis time for each event that will happen for a service and the analysis frequency for each service as well as the probability that a problem will happen for a service. Moreover, in order to find the optimal number of planners, it is essential to know the probability that an action will be chosen for a service and the planning time for each action for a service. Furthermore, we need to know the execution time that is taken by an executor to carry out each adaptation action over each service in order to determine the optimal number of executors.

In order to do all of that, it is essential to know an estimate of the execution time of each service as well as an estimate of the data transfer time between services. For that, we opt for timed SBPs. In this work, we assume that there is maximum Quality of Service (QoS) degradation time that a timed SBP can not exceed.

4.3.2 Timed SBP Modeling

As the structure of a timed SBP can be mapped to a graph, we choose graph theory to represent such SBP as a directed graph called the *timed SBP graph* according to the following definition.

Definition 4.3.1 (Timed SBP graph)**.** *A timed SBP graph* $G = (V, E, I, \tau, \varepsilon, \delta, \iota)$ *is a directed graph where:*

- V *is the set of nodes;*
- $E \subseteq V \times V$ *is the set of edges that represents the data dependencies between nodes, such that* $(v_i, v_j) \in E$ *if the output data of node* i *is required for the execution of node* j*;*
- $I \subset V$ *is the set of initial nodes;*
- $\tau : V \leftarrow \Gamma$ *is a function that maps nodes to their types;*
- $\varepsilon : V \leftarrow \mathbb{N}$ *is a function that assigns, for each service* $v \in V$*, where* $\tau(v) = service$*, an estimated execution time;*
- $\delta : V \times V \leftarrow \mathbb{N}$ *is a function that assigns, for each pair of services, an estimated data transfer time between them;*
- $\iota : V \leftarrow \mathbb{N}$ *is a function that maps each node* $v \in V$ *to a unique identifier* $id \in \mathbb{N}$*.*

Based on Definition 4.3.1, Figure 4.2 shows an example of a timed SBP graph. The number within the dash circle represents the estimated execution time of a service. The number next to an edge represents an inter-service estimated data transfer time. For example, the estimated execution time of *Choose supplier* is equal to 16, while the estimated data transfer time between the latter and *Contact supplier 1* (respectively Contact supplier 2, Contact supplier 3) is equal to 11, and it is equal to 9 between *Contact supplier 1* and *Receive products*.

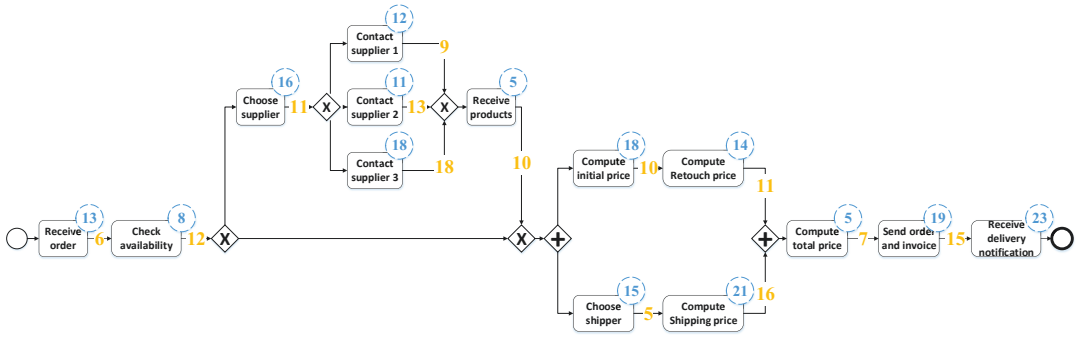


Figure 4.2: Example of timed SBP graph.

4.3.3 Notations and Assumptions

Let S be the set of services that make up a timed SBP and PS the sets of parallel services. Let M, A, P and E respectively represent the set of candidate monitors, analyzers, planners and executors that may be assigned to S. Each set has as cardinality the number of services, *i.e.* $|S|$, since in the worst case different monitors, analyzers, planners and executors will be assigned to the autonomic management of services.

Let mt_s be the monitoring time for service $s \in S$. It is equal to the sum of the time taken by a monitor to collect monitoring data from s and the time needed to save data in the knowledge base and the time needed to send an event to an analyzer component. Let mf_s be the monitoring frequency for service s (*e.g.* monitor each 5 ms) and I_s the set of time stamps at which we can monitor it, where the step size between each element in I_s and the next element is equal to mf_s.

We define Ev_s as the set of events that could result from the analysis of the monitoring data collected from service s. It contains the event 'no problem found' in addition to all the predefined problems that could happen. Let at_{ev_s} be the analysis time for event $ev \in Ev_s$ when it happens for service s. It is equal to the sum of the time taken by an analyzer to identify that event ev happened for s and the time needed to send an alert to a planner component when an adaptation is required. Let af_s be the analysis frequency for service s (*e.g.* analyze each 10 ms) and J_s the set of time stamps at which we can analyze it, where the step size between each element in J_s and the next element is equal to af_s. We denote by p_{ev_s} the probability that problem ev will happen for service s. Thus, $\forall s \in S, \sum_{ev_s \in Ev_s} p_{ev_s} = 1$.

We also define Ac_s as the set of adaptation actions that can be produced by a planner component in order to process problems that may happen for service s. Let pt_{ac_s} be the planning time for action $ac \in Ac_s$ for service s. It is equal to the sum of the time taken by a planner to generate the adaptation plan $ac \in Ac_s$ for s and the time needed to send management actions to an executor component. We denote by p_{ac_s} the probability that action ac will be chosen for service s. Hence, $\forall s \in S, \sum_{ac_s \in Ac_s} p_{ac_s} = 1$. Let et_{ac_s} be the execution time that is taken by an executor to carry out the adaptation action ac

over service s. Note that a degradation of QoS lasts the time taken to produce and carry out adaptation plans over SBP services. We define mdt as the maximum allowed QoS degradation time for a timed SBP. It keeps the QoS degradation below a certain upper limit.

Without loss of generality, we assume that all the monitors, analyzers, planners and executors have the same requirements in terms of hardware resources (RAM, CPU, etc.), so that the optimization of their hosting cost will be reduced to the optimization of their number.

4.3.4 Problem Formulation

In this formulation, we consider the following decision variables: α_m indicates if monitor $m \in M$ is used by at least one service; β_a indicates if analyzer $a \in A$ is used by at least one monitor; σ_p indicates if planner $p \in P$ is used by at least one analyzer; φ_e indicates if executor $e \in E$ is used by at least one planner; y_m^a specifies if monitor $m \in M$ is linked to analyzer $a \in A$; z_a^p specifies if analyzer $a \in A$ is linked to planner $p \in P$; w_p^e specifies if planner $p \in P$ is linked to executor $e \in E$; x_m^s expresses if monitor $m \in M$ is assigned to service $s \in S$; γ_{sm}^t expresses if monitor $m \in M$ is assigned to service $s \in S$ at time $t \in T$; λ_{sa}^t expresses if analyzer $a \in A$ is assigned to service $s \in S$ at time $t \in T$. These decision variables are equal to 1 if their indications are verified and 0 otherwise.

Given the aforementioned assumptions and notations, our problem can be formulated in equations [4.1-4.14] where objective function 4.1 aims to minimize the number of monitors, analyzers, planners and executors for the management of a timed SBP. Constraint 4.2 makes sure that only one monitor is assigned to each service all along the SBP life cycle. Constraint 4.4 (respectively 4.6, 4.8) makes sure that if a monitor is used (respectively analyzer, planner), then it is linked to only one analyzer (respectively planner, executor). Constraint 4.3 ensures that if a monitor is assigned to at least one service, then it is considered as used. Constraint 4.5 (respectively 4.7, 4.9) ensures that if an analyzer (respectively planner, executor) has at least one monitor (respectively analyzer, planner) linked to it, then it is considered as used. Constraints [4.10-4.14] guarantee that each component is assigned to at most one service at a time. The maximum allowed QoS degradation time for a given SBP is met according to constraint 4.14.

$$min \sum_{m \in M} \alpha_m + \sum_{a \in A} \beta_a + \sum_{p \in P} \sigma_p + \sum_{e \in E} \varphi_e \tag{4.1}$$

Subject to:

$$\sum_{m \in M} x_m^s = 1 \qquad \forall s \in S \qquad (4.2)$$

$$x_m^s \leqslant \alpha_m \qquad \forall s \in S,\ \forall m \in M \qquad (4.3)$$

$$\sum_{a \in A} y_m^a = \alpha_m \qquad \forall m \in M \qquad (4.4)$$

$$y_m^a \leqslant \beta_a \qquad \forall m \in M,\ \forall a \in A \qquad (4.5)$$

$$\sum_{p \in P} z_a^p = \beta_a \qquad \forall a \in A \qquad (4.6)$$

$$z_a^p \leqslant \sigma_p \qquad \forall a \in A,\ \forall p \in P \qquad (4.7)$$

$$\sum_{e \in E} w_p^e = \sigma_p \qquad \forall p \in P \qquad (4.8)$$

$$w_p^e \leqslant \varphi_e \qquad \forall p \in P,\ \forall e \in E \qquad (4.9)$$

$$x_m^s \leqslant \gamma_{sm}^t \qquad \forall m \in M,\ \forall s \in S,\ \forall i \in I_s,\ \forall t \in \{i, ..., i + mt_s - 1\} \qquad (4.10)$$

$$\sum_{s \in S} \gamma_{sm}^t \leqslant 1 \qquad \forall m \in M,\ \forall t \in T \qquad (4.11)$$

$$\sum_{m \in M} \sum_{ev_s \in Ev_s} p_{ev_s}.x_m^s.y_m^a \leqslant \lambda_{sa}^t \qquad \forall a \in A,\ \forall s \in S,\ \forall j \in J_s,\ \forall t \in \{j, ..., j + at_{ev_s} - 1\} \qquad (4.12)$$

$$\sum_{s \in S} \lambda_{sa}^t \leqslant 1 \qquad \forall a \in A,\ \forall t \in T \qquad (4.13)$$

$$\sum_{S \in PS} \sum_{m \in M} \sum_{a \in A} \max_{p \in P} \Big(\sum_{s \in S} \sum_{ac_s \in Ac_s} p_{ac_s}.pt_{ac_s}.x_m^s.y_m^a.z_a^p \Big) + \max_{e \in E} \Big(\sum_{s \in S} \sum_{ac_s \in Ac_s} p_{ac_s}.et_{ac_s}.x_m^s.y_m^a.z_a^p.w_p^e \Big) \leqslant mdt \qquad (4.14)$$

However, the above model is not linear due to constraints 4.12 and 4.14. In order to linearize it, we introduce new decision variables u_a^s, v_p^s, and k_e^s defined as follows:

$$u_a^s = x_m^s.y_m^a \qquad \forall s \in S,\ \forall m \in M,\ \forall a \in A \qquad (4.15)$$

$$v_p^s = x_m^s.y_m^a.z_a^p \qquad \forall s \in S,\ \forall m \in M,\ \forall a \in A,\ \forall p \in P \qquad (4.16)$$

$$k_e^s = x_m^s.y_m^a.z_a^p.w_p^e \qquad \forall s \in S,\ \forall m \in M,\ \forall a \in A,\ \forall p \in P,\ \forall e \in E \qquad (4.17)$$

We substitute equation 4.15 in constraint 4.12 and equations 4.16 and 4.17 in constraint 4.14. Constraints 4.12 and 4.14 respectively become as follows:

$$\sum_{ev_s \in Ev_s} p_{ev_s}.u_a^s \leqslant \lambda_{sa}^t \quad \forall a \in A,\ \forall s \in S, \forall j \in J_s,\ \forall t \in \{j, ..., j + at_{ev_s} - 1\} \qquad (4.18)$$

$$\sum_{S \in PS} \max_{p \in P}(\sum_{s \in S} \sum_{ac_s \in Ac_s} p_{ac_s}.pt_{ac_s}.v_p^s) + \max_{e \in E}(\sum_{s \in S} \sum_{ac_s \in Ac_s} p_{ac_s}.et_{ac_s}.k_e^s) \leqslant mdt \quad (4.19)$$

Then we must add the following logical constraints:

$$x_m^s + y_m^a \leqslant u_a^s + 1 \qquad \forall s \in S,\ \forall m \in M,\ \forall a \in A \quad (4.20)$$

$$x_m^s + y_m^a + z_a^p \leqslant v_p^s + 2 \qquad \forall s \in S,\ \forall m \in M,\ \forall a \in A,\ \forall p \in P \quad (4.21)$$

$$x_m^s + y_m^a + z_a^p + w_p^e \leqslant k_e^s + 3 \quad \forall s \in S,\ \forall m \in M,\ \forall a \in A,\ \forall p \in P,\ \forall e \in E \quad (4.22)$$

Considering the number of deployments of applications in a public Cloud (see the previous chapter, Section 3.2.4 for more details), and as we will show in the evaluation section (Section 4.6), the time needed to solve the optimization model using the CPLEX solver is not acceptable. It can exceed two hours; and whenever the number of services goes beyond 26, the solver can not find the optimal solution. Consequently, finding the optimal solution, in this context, is not possible. Therefore, to tackle this NP-hard problem [80], we propose in the following section an approach that provides near-optimal solutions in a polynomial time.

4.4 Proposed Approach

4.4.1 Approach Overview

In this section, we introduce an approach based on two algorithms called ParalelServices and M-A-P-E_Assignment, for the determination of the appropriate number of monitors, analyzers, planners and executors that will be used by SBP services for their management such that this number is minimized while avoiding management bottlenecks.

The ParalelServices algorithm, whose pseudo-code is given in Algorithm 4.1, takes as input a timed SBP graph. It aims to determine all sets of services that can run in parallel.

The M-A-P-E_Assignment algorithm, whose pseudo-code is given in Algorithm 4.2, takes as input the sets of parallel services as well as the data needed for managing each service. It consists in assigning AM components to services while fulfilling the constraints given by equations [4.1-4.14].

4.4.2 ParallelServices Algorithm

First of all, *CurrentParallelSet* and *CurrentPendingSet* are initialized to the initial services of the timed SBP graph and the empty set, respectively (*cf.* lines 1-2). They are respectively used to store the current set of services that can run in parallel and the current set of pending services that will be executed after receiving the data required for their execution. After that, the current set of parallel services is added to the resulting sets (*cf.* line 3), and an array is initialized with the estimated execution times of services (*cf.*

lines 4-5). PARALELSERVICES iterates until determining all the sets of parallel services (*cf.* line 6 and line 28). To do so, the elements of the current sets are assigned to the previous sets, and the current sets are initialized to the empty set (*cf.* lines 7-9). Then the algorithm determines the minimum time from the times required by the current services to complete or start execution in order to determine the new sets of parallel and pending services (*cf.* line 10). First, it iterates over the previous set of parallel services; and for each service v_i, the algorithm checks whether this service has completed its execution; i.e., the remaining time to complete its execution is equal to zero (*cf.* lines 11-13). As a result, for each successor, whether the type of this candidate node is either an *AND/OR -Join* and all its predecessors have completed their execution or not an *AND/OR -Join*, if the current node is of type *service*, then it is added to the current set of pending services; otherwise, PARALELSERVICES continues to explore the successors of the candidate node seeking a node of *service* type (*cf.* lines 13-17). If, the current has not completed its execution, it is added to the current set of parallel services (*cf.* lines 18-19). Second, the algorithm iterates over the previous set of pending services; and for each service v_i, it checks whether the data transfer time to it has been completed (*cf.* lines 20-21). If it is the case, v_i is added to the current set of parallel services (*cf.* lines 22-23), otherwise it is added to the current set of pending services (*cf.* lines 24-25). Afterwards, if the current set of parallel services has not been already determined in a previous iteration, it is added

to the resulting sets of parallel services (*cf.* lines 26-27).

Algorithm 4.1: ParallelServices

```
   Data:  - G = (V, E, I, τ, ε, δ, ι): SBP graph
   Result: - ParallelSets : Set of sets of parallel services
   begin
1      CurrentParallelSet ← Services(I);
2      CurrentPendingSet ← ∅;
3      ParallelSets ← {CurrentParallelSet};
4      for s ∈ S do
5          t[s] ← ε(s);
6      repeat
7          PreviousParallelSet ← CurrentParallelSet;
8          PreviousPendingSet ← CurrentPendingSet;
9          CurrentParallelSet, CurrentPendingSet ← ∅;
10         time ← MinimumTime(PreviousParallelSet, PreviousPendingSet);
11         for v_i ∈ PreviousParallelSet do
12             t[v_i] ← t[v_i] − time;
13             if t[v_i] = 0 then
14                 for (v_i,v_j) ∈ E do
15                     if (τ(v_j) ∉ {AND-Join, OR-Join}) or (τ(v_j) ∈ {AND-Join,
                           OR-Join} and AllPredecessors(v_j)) then
16                         CurrentPendingSet ← CurrentPendingSet ∪
                             {Services(v_j)};
17                         dt[v_j] ← δ(v_i,v_j);
18             else
19                 CurrentParallelSet ← CurrentParallelSet ∪ {v_i};
20         for v_i ∈ PreviousPendingSet do
21             dt[v_i] ← dt[v_i] − time;
22             if dt[v_i] = 0 then
23                 CurrentParallelSet ← CurrentParallelSet ∪ {v_i};
24             else
25                 CurrentPendingSet ← CurrentPendingSet ∪ {v_i};
26         if ∄ set s.t.(set ∈ ParallelSets and CurrentParallelSet ⊆ set) then
27             ParallelSets ← ParallelSets ∪ CurrentParallelSet;
28     until (CurrentParallelSet = ∅ and CurrentPendingSet = ∅);
```

4.4.3 M-A-P-E_Assignment Algorithm

M-A-P-E_Assignment is mainly based on two procedures, called M/A_Assignment and P-E_Assignment. The M/A_Assignment procedure, whose pseudo-code is given in Algorithm 4.6, aims to assign a minimum number of monitors and analyzers to the SBP services. The P-E_Assignment procedure, whose pseudo-code is given in

Algorithm 4.3, tries to assign a minimum number of planners and executors to the services such that the maximum allowed QoS degradation time for the SBP is not exceeded. To avoid management bottlenecks, our algorithms ensure the assignment of different monitors (respectively analyzers, planners, and executors) to parallel services.

Since each monitor must be linked to only one analyzer and multiple monitors can be linked to the same analyzer, the number of monitors that will be used to manage a timed SBP must be greater than or equal to the number of analyzers, *i.e.* $M \geq A$. In addition, each analyzer (respectively planner) must be linked to only one planner (respectively executor), and multiple analyzers (respectively planners) can be linked to the same planner (respectively executor). The number of analyzers (respectively planners) must be greater than or equal to the number of planners (respectively executors), *i.e.* $A \geq P$ (respectively $P \geq E$). Therefore, we must have $M \geq A \geq P \geq E$, so $P + E \leq 2 \times A$.

As $P + E \leq 2 \times A$, if we start by determining the number of monitors and analyzers, the number of planners and executors may be less than the minimum required number of planners and executors that maintain the QoS degradation time for SBP below a certain upper limit. Consequently, M-A-P-E_ASSIGNMENT starts by determining the number of planners and executors by invoking the P-E_ASSIGNMENT procedure (*cf.* line 1). After that, it determines the number of analyzers and monitors by applying the M/A_ASSIGNMENT procedure twice (*cf.* lines 2-3).

Algorithm 4.2: M-A-P-E_ASSIGNMENT

Data: - ParallelSets: Set of sets of parallel services
- T: Array containing the required execution time of each service
- MT, MF: Arrays containing, respectively, the monitoring time and the monitoring frequency for each service
- AT, AF: Arrays containing, respectively, the analysis time and the analysis frequency for each service
- PT: Array containing the maximum planning time for each service
- ET: Array containing the maximum adaptation time for each service
- mdt: Maximum allowed QoS degradation time for the SBP

Result: - M, A, P, E: Arrays containing, respectively, the monitor, analyzer, planner, and executor assigned to each service
- n_m, n_a, n_p, n_e: Number of, respectively, monitors, analyzers, planners, and executors

begin

1 P, n_p, E, $n_e \leftarrow$ P-E_ASSIGNMENT(PT, ET, mdt);

2 A, $n_a \leftarrow$ M/A_ASSIGNMENT($PSets$, AT, AF);
/*PSets is a set of sets of services where services belonging to a set are managed by the same Planner*/

3 M, $n_m \leftarrow$ M/A_ASSIGNMENT($ASets$, MT, MF);
/*ASets is a set of sets of services where services belonging to a set are managed by the same Analyzer*/

4.4.3.1 P-E_Assignment procedure

Initially, the P-E_ASSIGNMENT procedure dedicates a planner and an executor for the management of the given SBP (*cf.* lines 2-5). The total QoS degradation time for SBP is then calculated, which is equal to the sum of time required to generate and execute adaptation plans over the services that make up SBP (*cf.* line 5). The algorithm iterates as long as the total QoS degradation time for SBP exceeds the maximum allowed QoS degradation time (*cf.* line 6). It operates in two steps: First, it adds a new planner by invoking the P_ASSIGNMENT procedure (*cf.* lines 7-8). Second, if the maximum allowed QoS degradation time is still not respected (*cf.* line 9), it calculates the new QoS degradation time whether we add: (i) a new planner by invoking the P_ASSIGNMENT procedure, and (ii) a new executor by invoking the E_ASSIGNMENT procedure (*cf.* lines 10-11). Then the algorithm adds the component (planner or executor) that further reduces the total QoS degradation time for SBP (*cf.* lines 12-20). The first step of the algorithm consists in ensuring that the number of planners is greater than or equal to the number of executors in order not to push any planner to link to more than one executor.

Algorithm 4.3: P-E_ASSIGNMENT procedure

```
   Data: mdt
   Result: P, n_p, E, n_e
   begin
1      tdt ← 0; /*tdt is the total QoS degradation time for the SBP*/
2      n_p, n_e ← 1;
3      for s ∈ S do
4          P[s], E[s] ← 1;
5          tdt ← tdt + PT[s] + ET[s];
6      while (tdt > mdt) do
7          n_p ← n_p + 1;
8          tdt, P ← P_ASSIGNMENT(n_p);
9          if tdt > mdt then
10             tdt_p, I ← P_ASSIGNMENT(n_p + 1);
11             tdt_e, set ← E_ASSIGNMENT(n_e + 1, tdt);
12             if (tdt_p < tdt_e) then
13                 n_p ← n_p + 1;
14                 tdt ← tdt_p;
15                 P ← I;
16             else
17                 n_e ← n_e + 1;
18                 tdt ← tdt_e;
19                 for s ∈ set do
20                     E[s] ← n_e;
```

P_Assignment procedure The main objective of the P_ASSIGNMENT procedure, whose pseudo-code is given in Algorithm 4.4, is to find the appropriate assignment of a given number of planners, *i.e.* n_p, to the SBP services such that the total QoS degradation time for SBP is minimized. It goes through the sets of parallel services set by set; and for each one, their elements are sorted in a decreasing order to promote services with maximum planning time (*cf.* lines 1-2). After that, different planners are assigned to the first n_p services in the current set; and for each planner assigned to a service, its working time is set to the planning time of the latter service (*cf.* lines 3-8). Then the total time required to generate adaptation plans for services belonging to the current set is calculated, and the latter is evenly shared between the n_p planners to further reduce the total QoS degradation time for SBP (*cf.* line 9). The total QoS degradation time is equal to the sum of the maximum time from the times taken by the planners to generate adaptation plans for each set of parallel services and the the sum of the maximum time from the times taken by the executors to carry out adaptation plans for each set of parallel services (it is given in equation 4.9 (left)). Afterwards, the algorithm iterates over the set of services to which no planner has been assigned (*cf.* line 10); and for each one, it starts with the first planner (*cf.* line 11), and it checks whether the latter might be assigned to the current service, that is whether if the sum of the working time of this planner and the planning time required for the current service is less than or equal to the estimated time that a planner is expected to take to generate adaptation plans for the current set of parallel services (*cf.* line 16). If it is the case, this planner is assigned to the current service and its working time is updated (*cf.* lines 17-19). Otherwise, the algorithm just updates the current planner (*cf.* line 20). In the case where the current planner number is greater than n_p, P_ASSIGNMENT looks for the appropriate planner that has the minimum working time to assign it to the current service (*cf.* lines 14-15). This step is repeated until a planner is assigned to the service (*cf.* lines 12-13). Finally,

the total QoS degradation time for SBP is calculated (*cf.* line 21).

Algorithm 4.4: P_ASSIGNMENT procedure

```
   Data:   - n_p
   Result: - P: Array containing the planner assigned to each service
           - tdt: Total QoS degradation time for the SBP
   begin
1      for set ∈ ParallelSets do
2          SortDecreasingOrderOfPlanningTimeOfServices(set);
3          currentP ← 1;
4          for s ∈ set where currentP ⩽ n_p do
5              P[s] ← currentP;
6              set ← set ∖ {s};
7              currentP ← currentP + 1;
8              timeP[currentP] ← PT[s];
9          time ← TotalPlanningTime(set)/n_p;
10         for s ∈ set do
11             currentP ← 1;
12             isAssigned ← false;
13             while not(isAdded) do
14                 if currentP > n_p then
15                     currentP ← GetPlanner(timeP);
16                 if timeP[currentP] + PT[s] ⩽ time or currentP > n_p then
17                     isAssigned ← true;
18                     P[s] ← currentP;
19                     timeP[currentP] ← timeP[currentP] + PT[s];
20                 currentP ← currentP + 1;
21     tdt ← QoSDegradationTime();
```

E_Assignment procedure The main purpose of the E_ASSIGNMENT procedure, whose pseudo-code is given in Algorithm 4.5, is to find the appropriate assignment of a new executor to the SBP services such that the total QoS degradation time for SBP is minimized. It starts by initializing the set of services to which the new executor will be assigned to the empty set (*cf.* line 1). Next, it goes through the sets of services, where the services belonging to a set are managed by the same planner, set by set, to ensure that each planner is linked to only one executor (*cf.* line 6). The set of services that further reduces the total QoS degradation time for SBP when the new executor is assigned to them is determined through the invocation of the QoSDegradationTime function (*cf.* lines 7-18). After that, the algorithm looks for other services to which it assigns the executor, and it comes to an end when it can not further reduce the total QoS degradation time

(*cf.* line 19).

Algorithm 4.5: E_ASSIGNMENT procedure

```
   Data: n_e, tdt
   Result: tdt, Services
   begin
1      Services ← ∅;
2      mintdt ← tdt;
3      repeat
4          tdt ← mintdt;
5          mintdt ← +∞;
6          for set ∈ PSets do
7              I ← E;
8              for s ∈ set do
9                  E[s] ← n_e; /*n_e is the new executor number*/
10             tdt ← QoSDegradationTime();
11             E ← I;
12             if tdt < mindt then
13                 mintdt ← tdt;
14                 BestSet ← set;
15         if mintdt < tdt then
16             for s ∈ BestSet do
17                 E[s] ← n_e;
18             Services ← Services ∪ BestSet;
19     until mintdt ⩾ tdt;
```

4.4.3.2 M/A_Assignment procedure

The M/A_ASSIGNMENT procedure, whose pseudo-code is given in Algorithm 4.6, aims to assign a minimum number of analyzers (respectively monitors) to the SBP services. It goes through the sets of services, where the services belonging to a set are managed by the same planner (respectively analyzer), set by set, to ensure that each analyzer (respectively monitor) is linked to only one planner (respectively analyzer) (*cf.* line 2). Then for each service, the algorithm looks for an analyzer (respectively monitor) which is not used by any service during any time period when the current service uses it to guarantee that each analyzer (respectively monitor) is assigned to only one service at a time (*cf.* lines 4-12). The latter analyzer (respectively monitor) is then assigned to the service (*cf.* line 13), and it is considered as used at these time periods (*cf.* lines 14-16). The number of analyzers

(respectively monitors) that will be used by services is possibly updated (*cf.* lines 17-18).

Algorithm 4.6: M/A_ASSIGNMENT procedure

```
   Data: Sets, AT, AF
   Result: n_c, C
   begin
1      n_c ← 0;
2      for set ∈ Sets do
3          for s ∈ set do
4              currentC ← 0;
5              repeat
6                  unoccupied ← true;
7                  currentC ← currentC + 1;
8                  for (i ← st_s; i < st_s + T[s]; i ← i + AF[s]) do
9                      for (t ← i; t < i + AT[s]; t ← t + 1) do
10                         if currentC ∈ occupied[t] then
11                             unoccupied ← false;
12             until unoccupied;
13             C[s] ← currentC;
14             for (i ← st_s; i < st_s + T[s]; i ← i + AF[s]) do
15                 for (t ← i; t < i + AT[s]; t ← t + 1) do
16                     occupied[t] ← occupied[t] ∪ {currentC};
17             if currentC > n_c then
18                 n_c ← currentC;
```

4.4.4 Theoretical Complexity

Our approach consists of two main algorithms called PARALELSERVICES and M-A-P-E_ASSIGNMENT. The first one consists in determining all the sets of parallel services (Algorithm 4.1). The worst case time complexity of this algorithm is bounded by $O(n^4)$, where n is the number of services that compose the given SBP.

The second algorithm aims to assign a minimum number of monitors, analyzers, planners, and executors to the SBP services such that the total QoS degradation time for SBP is maintained below a certain upper limit while avoiding management bottlenecks (Algorithm 4.2). It consists of two procedures, called P-E_ASSIGNMENT (Algorithm 4.3) and M/A_ASSIGNMENT (Algorithm 4.6):

- The complexity of the P-E_ASSIGNMENT procedure depends on those of the two invoked procedures, namely P_ASSIGNMENT and E_ASSIGNMENT, which have, respectively, a complexity of $O(n^4)$ and $O(n^4)$, where n is the number of services. The number of iterations of the while loop of P-E_ASSIGNMENT is equal to $2 \times n$. Hence, its overall complexity is $O(2 \times n \times n^4) \simeq O(n^5)$;

- The complexity of the M/A_ASSIGNMENT procedure is $O(n^3 \times m^2)$, where n is the number of services and m is the maximum execution time from the estimated execution times of the SBP services. The number of iterations of the repeat loop is equal to n.

The worst case time complexity of the M-A-P-E_ASSIGNMENT algorithm is $O(n^5 + n^3 \times m^2)$. In summary, we can say that the theoretical complexity of the proposed approach is bounded by $O(n^5 + n^3 \times m^2)$.

4.5 Illustrative Example

Herein, we present an example illustrating how the approach works. Let us consider the example depicted in Figure 4.3 that represents a timed SBP graph.

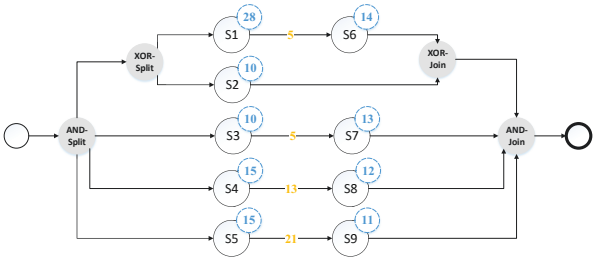


Figure 4.3: Example of timed SBP graph.

Applying the PARALLELSERVICES algorithm, as shown in Figure 4.4, the first iteration considers the initial sets: *ParallelSet*={S1,S2,S3,S4,S5} and *PendingSet*=$\emptyset$. The number within the blue rectangle represents the estimated remaining time required by a service to complete its execution, while the number within the yellow rectangle represents the estimated remaining time required by a service to start its execution. The minimum time is equal to 10, so the estimated remaining time required by the service $S1$, $S2$, $S3$, $S4$ and $S5$ to complete their execution is 18, 0, 0, 5 and 5, respectively. It is equal to 0 for services $S2$ and $S3$, then their successors are added to the current pending set, *i.e.* *PendingSet*=$\{S7\}$ and *ParallelSet*=$\{S1, S4, S5\}$. In the second iteration, the minimum time is equal to 5. Thus, the estimated remaining time required by service $S1$, $S4$ and $S5$ to complete their execution is 13, 0 and 0, respectively; and the estimated remaining time required by service $S7$ to start its execution is 0. It is equal to 0 for services $S4$, $S5$ and $S7$. The successors of services $S4$ and $S5$ ($S8$ and $S9$) are added to the current set of pending services, where the estimated time required by each one to start its execution is mentioned within the yellow rectangle; and service $S7$ is added to the current set of parallel services since the time required to transfer data from $S3$ to $S7$ is completed, where the time required by $S7$ to complete its execution is mentioned within the blue rectangle. The same process is applied until reaching the 7^{th} iteration with *PendingSet*=$\emptyset$ and *ParallelSet*=$\emptyset$.

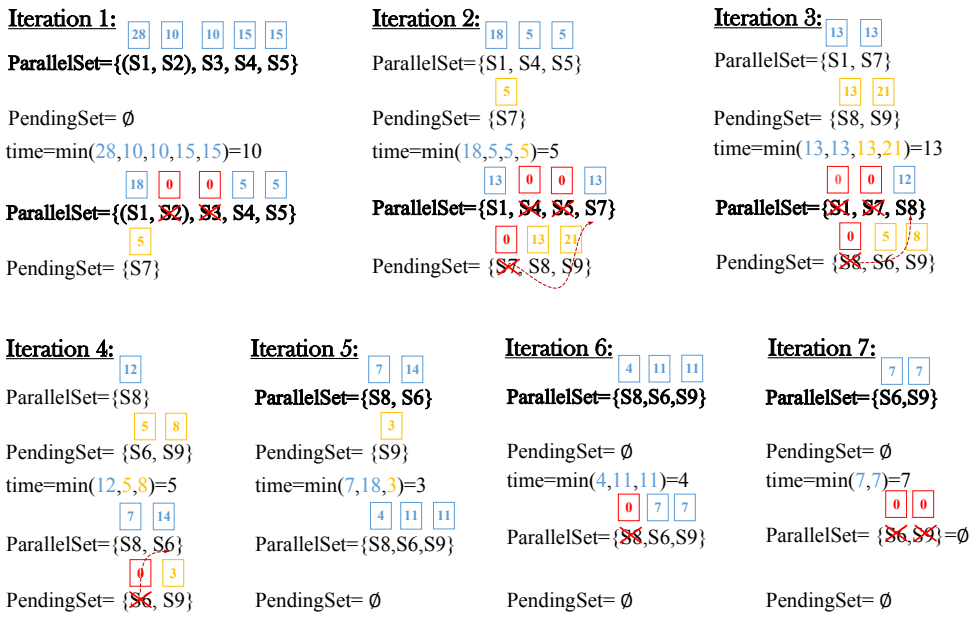


Figure 4.4: Execution trace of PARALLELSERVICES algorithm.

The PARALLELSERVICES algorithm comes to an end after seven iterations and checking that all the sets of parallel services are determined. Thus, it yields the following output:
ParallelSets=$\{\{(S1, S2), S3, S4, S5\}, \{S1, S7\}, \{S6, S8, S9\}\}$

Then the P-E_ASSIGNMENT algorithm is applied to determine the appropriate number of planners and executors for the management of the given timed SBP using the following inputs:

- **ParallelSets**=$\{\{(S1, S2), S3, S4, S5\}, \{S1, S7\}, \{S6, S8, S9\}\}$;
- **PT**=$[3, 2, 4, 3, 2, 1, 2, 2, 3]$;
- **ET**=$[3, 2, 2, 3, 3, 1, 2, 3, 2]$;
- **mdt**=26.

As shown in Figure 4.5, the first iteration considers a planner and an executor for the management of the given SBP. The time taken by the planner (respectively executor) to manage the services that belong to the first set of parallel services $\{(S1,S2),S3,S4,S5\}$ is equal to $max(3,2)+4+3+2=12$ (respectively $max(3,2)+2+3+3=11$). In fact, services $S1$ and $S2$ are two alternative services, then we take the maximum time between the times needed to generate adaptation plans for these services (*i.e.* $max(3,2)$); and as a planner (respectively executor) must be assigned to only one service at a time, it sequentially produces adaptation plans for the current services. As a result, we have done

the sum for the planning times (respectively carry out times). However, the time taken by the planner (respectively executor) to manage the services that belong, respectively, to the second and the third set of parallel services is equal to $3 + 2$ (respectively $3 + 2$) and $1 + 2 + 3$ (respectively $1 + 3 + 2$). Using one planner and one executor, the total QoS degradation time for SBP is equal to 45. It is greater than the maximal allowed QoS degradation time that is equal to 26, so a new planner is added. In the second iteration, for each set of parallel services, the total time required for generating adaptation plans for the current services is calculated, and it is evenly shared between the two planners. The first set of parallel services consists of services $S1$, $S2$, $S4$, $S3$ and $S5$, so planner number 1 is assigned to services $S1$, $S2$ and $S4$, and planner number 2 is assigned to the services $S3$ and $S5$. The same process is applied to the second and third sets: planner number 1 is assigned to services $S6$ and $S8$, and planner number 2 is assigned to services $S7$ and $S9$. Using tow planners and one executor, *tdt* is equal to 34. It is still greater than the maximal allowed QoS degradation time. In this case, we calculate the new *tdt* whether we add (i) a new planner, or (ii) a new executor. It is respectively equal to 33 and 25. A new executor is then added for the management of SBP because it further reduces *tdt* and the algorithm comes to an end ($25 \leqslant 26$).

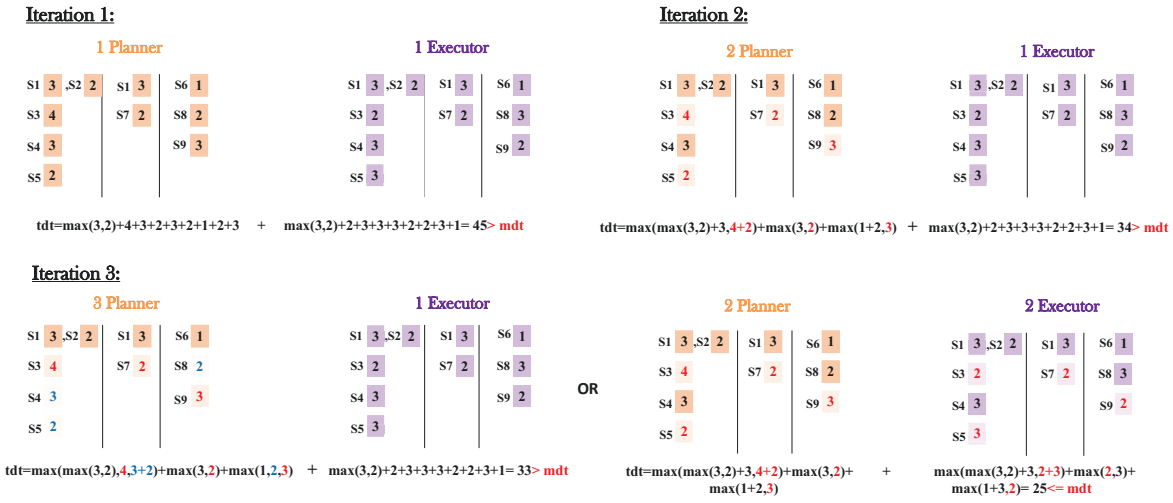


Figure 4.5: Execution trace of P-E_ASSIGNMENT algorithm.

The P-E_ASSIGNMENT algorithm comes to an end after three iterations. It yields the following outputs:

P	1	1	2	1	2	1	2	1	2

After that, the M/A_ASSIGNMENT algorithm is applied to determine the appropriate number of analyzers using the following inputs:

E	1	1	2	1	2	1	2	1	2

- **PSets**=$\{\{S1, S2, S4, S6, S8\}, \{S3, S5, S7, S9\}\}$;
- **AT**=$[3, 1, 1, 2, 2, 1, 3, 2, 2]$;
- **AF**=$[7, 4, 4, 5, 5, 4, 4, 5, 3]$.

As shown in Figure 4.6 (right), the algorithm goes through the sets of services *PSets*, set by set, where the services belonging to a set are managed by the same planner; and for each service in the current set, it looks for an unoccupied analyzer. At first, analyzer number 1, *i.e.* $A1$, is assigned to service $S1$. Since $S1$ or $S2$ will be executed, $A1$ is also assigned to service $S2$. $A1$ is not occupied during any time period when service $S4$ will use it, i.e. [5,7](left), the latter is assigned to $S4$. The same process is applied to services $S6$ and $S8$. To ensure that each analyzer is linked to only one planner, different analyzers from the analyzers linked to planner number 1 must be used to manage the services to which planner number 2 is assigned, that is we can not use $A1$ to manage service $S3$, $S5$, $S7$ or $S9$. At first, analyzer number 2, *i.e.* $A2$, is assigned to service $S3$. We try then to assign $A2$ to service $S5$. In fact $A2$ is not occupied during any time period when service $S5$ will use it, *i.e.* [5,7],[10,12], it is assigned to $S5$. The same process is repeated for services $S7$ and $S9$, and $A2$ is assigned to them.

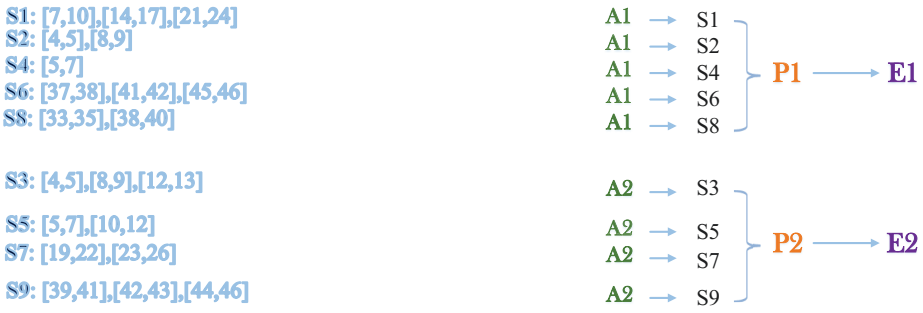


Figure 4.6: Execution trace of M/A_ASSIGNMENT algorithm: Assignement of analyzers to SBP services.

Finally, the M/A_ASSIGNMENT algorithm is applied in order to determine the appropriate number of monitors using the following inputs:

- **ASets**=$\{\{S1, S2, S4, S6, S8\}, \{S3, S5, S7, S9\}\}$;
- **MT**=$[3, 2, 1, 3, 2, 2, 4, 2, 3]$;
- **MF**=$[5, 3, 4, 3, 5, 3, 4, 3, 3]$.

As shown in Figure 4.7 (right), the algorithm goes through the sets of services, set by set, where the services belonging to a set are managed by the same analyzer; and for each service in the current set, it looks for an unoccupied monitor. At first, monitor number 1, *i.e.* $M1$, is assigned to service $S1$. Since $S1$ or $S2$ will be executed, $M1$ is also assigned to service $S2$. $M1$ is occupied when service $S4$ will use it (it is already used by service $S2$ at the time periods [3,5] and [6,8]), then monitor number 2, *i.e.* $M2$, is assigned to $S4$. $M1$ is not occupied during any time period when service $S6$ will use it, *i.e.*[36,38],[39,41],[42,44],[45,47] (left), $M1$ is assigned to $S6$. We can not assign $M1$ to service $S8$ because it is occupied during the time period [37,38] when $S8$ will use it, but $M2$ is not occupied at these time periods: $M2$ is assigned to $S8$. To ensure that each monitor is linked to only one analyzer, different monitors from the monitors linked to analyzer number 1 must be used to manage the services to which analyzer number 2 is assigned. At first, monitor number 3, *i.e.* $M3$, is assigned to service $S3$. We try then to assign $M3$ to service $S5$. $M3$ is not occupied by any service when $S5$ will use it. It is also assigned to $S5$. The same process is repeated for services $S7$ and $S9$, and $M3$ is also assigned to them.

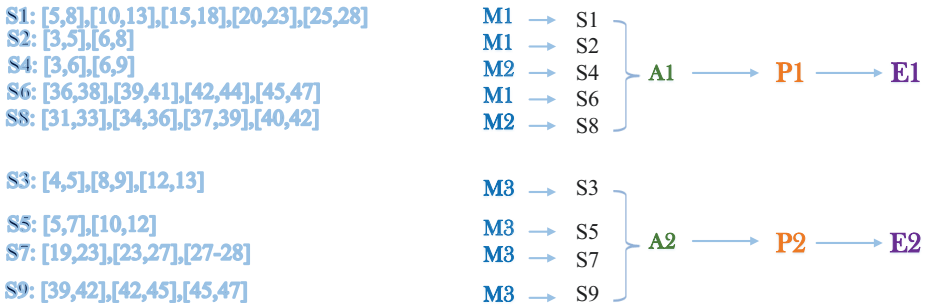


Figure 4.7: Execution trace of M/A_ASSIGNMENT algorithm: Assignement of monitors to SBP services.

4.6 Performance Evaluation

In this section, we present a series of performance evaluations of our approach in terms of execution time and quality of provided solutions. In order to show the efficiency of this approach, we compare it to our solution presented in the previous chapter, Section 3.3.1, to verify how the approach enhances the AM components optimization process for the management of timed SBPs. We also compare the solution provided by this approach with the solution obtained by solving the formal model presented in Section 4.3.4 using the CPLEX solver, when such a solution is possible (see the previous chapter, Section 3.4.3 for more details).

The different experiments are carried out on an Intel Core $i7$ PC with 2.70 GHz and

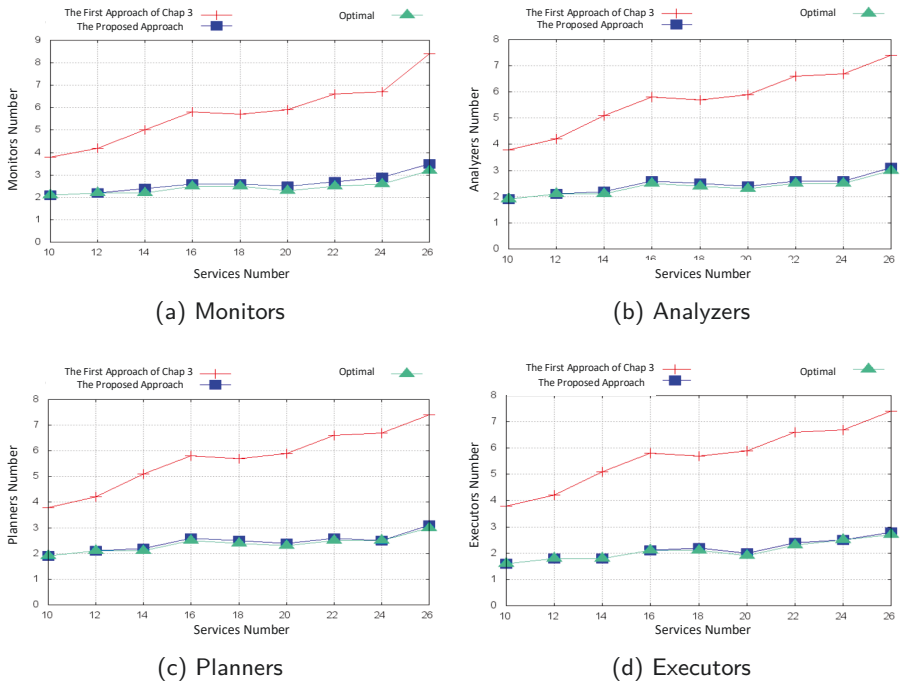
8GB of RAM. We use the commercial CPLEX solver 12.6 to solve the ILP formulation (Section 4.3.4).

In order to evaluate the performance of the proposed approach, we implement a generator of random timed SBPs. To the best of our knowledge, there is no available public dataset of timed business process models. Thus, to perform the experiments, we generate different connected timed SBP graphs with a number of services varying from 10 to 100, in increments of 10, and a number of edges ranging from $n-1$ to $3.2 \times (n-1)$ (*i.e.* 320% of $(n-1)$), in increments of 0.2, where n is the order of the graph. Indeed, we do not generate graphs with more edges, since whenever the number of edges goes beyond $2.4 \times (n-1)$, the lower bound number of AMs is equal to n (see the previous chapter for more details). All the results are average values across 10 independent runs. All data inputs are randomly generated and detailed in Table 4.1.

Information	**Range**
Number of services	$[10-100]$
Number of edges	$[(n-1)-3.2\times(n-1)]$
Execution time of a service	$[5-30]$
Data transfer time between services	$[5-30]$
Monitoring time	$[2-10]$
Monitoring frequency	$[1-5]$
Analysis time	$[2-10]$
Analysis frequency	$[1-5]$
Planning time	$[2-10]$
Adaptation time	$[2-10]$
Maximum allowed QoS degradation time	$[0-50]$

Table 4.1: Characteristics of data inputs of generated SBP graphs and AM components.

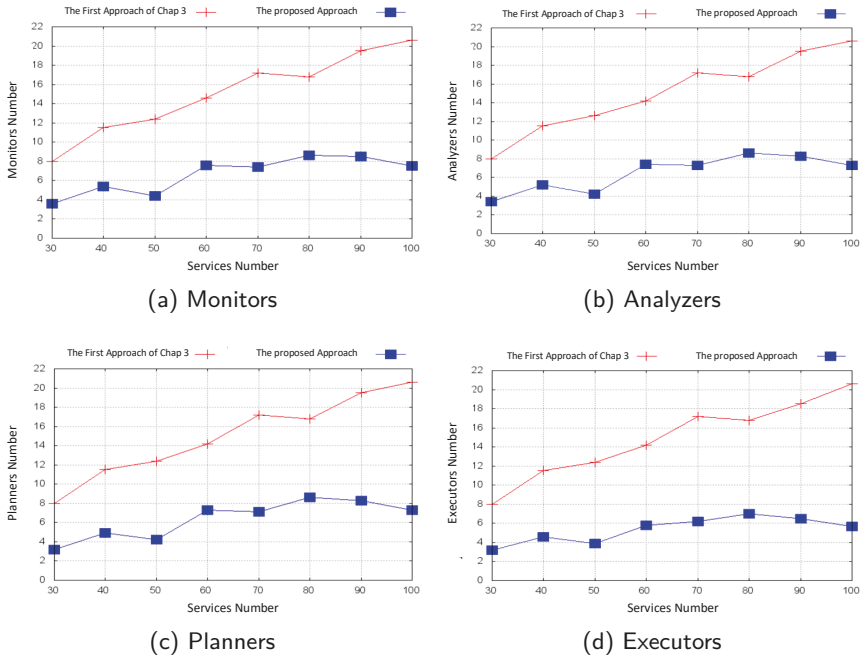
The experimental results on the randomly generated dataset are illustrated in Figures 4.8 and 4.9. Compared to our previous approach, it can be seen that the proposed approach outperforms the previous approach in terms of reducing the number of monitors, analyzers, planners and executors for the management of timed SBPs. It decreases the number of AMs by about 54%. On the other hand, the gap between our approximate solution and the optimal solution is very small and does not exceed 10.34%. It means that our solution is very close to the optimal solution. As for the execution time, the results presented in Figure 4.10 show that our proposed approach takes a very short time to provide a near-optimal solution (less than 0.07 second). However, the exact method, solved by the ILP solver, takes more than two hours for 26 services. Note that the solver can not to find the optimal solution for the SBP graphs with more than 26 services due to a problem of memory.



(a) Monitors (b) Analyzers

(c) Planners (d) Executors

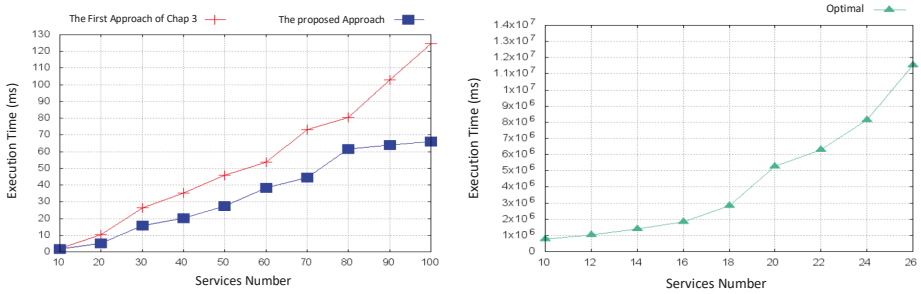
Figure 4.8: Number of monitors, analyzers, planners, and executors versus number of services varying from 10 to 26.



(a) Monitors (b) Analyzers

(c) Planners (d) Executors

Figure 4.9: Number of monitors, analyzers, planners, and executors versus number of services varying from 30 to 100.



(a) (b)

Figure 4.10: Execution time versus number of services.

The experimental results demonstrate that our approach outperforms the previous one significantly in terms of both quality and execution time.

4.7 Conclusion

In this chapter, we broke down AM into its main components (monitor, analyzer, planner, executor) to minimize the number of each one separately for the management of timed SBPs while avoiding management bottlenecks. We formally defined the problem as ILP. Since solving the ILP formulation is time-consuming even for small timed SBPs, we proposed a near-optimal approach to solve this problem in a polynomial time, and an illustrative example of how it works was given. The experimental results show the efficiency of our approach.

The performance of the management SBPs depend highly on the autonomic resources allocated to them and the placement decisions of these latter in the Cloud. One of the key challenges faced by Cloud providers is to optimize the placement cost of the autonomic resources that will be used for the management of SBPs while fulfilling SBP QoS requirements. Hence, in the next chapter, we will focus on how to formulate this problem and solve it in a polynomial time.

Chapter 5

Placement of Autonomic Managers for the management of SBPs in the Cloud

Contents

5.1 Introduction 77

5.2 Preliminaries: IaaS Cloud provider, Deployed SBP 77

5.2.1 IaaS Cloud Provider 77

5.2.2 Deployed SBP 78

5.3 Problem Description 79

5.4 Placement of AMs in the Cloud for Efficient Management of SBPs 79

5.4.1 Problem Formulation 80

5.4.2 Proposed Algorithm 81

5.4.3 Illustrative Example 82

5.4.4 Experimental Results 84

5.5 Joint Optimization of the Number of AMs and their Placement in the Cloud for Efficient Management of SBPs 86

5.5.1 Problem Formulation 86

5.5.2 Proposed Approach 91

5.5.3 Illustrative Example 95

5.5.4 Experimental Results 96

5.6 Conclusion 98

5.1 Introduction

The performance of SBPs depend highly on the Cloud resources allocated to them includes: (i) the allocation of the adequate number of autonomic resources for their management, which is the subject of our two previous chapters, and (ii) the allocation of the adequate Cloud resources to host and execute these AMs and the services that make up SBPs. Since several research works (such as [44, 45, 49, 48]) have treated the problem of the allocation of the adequate Cloud resources to SBP services, in this chapter, we focus on the allocation of Cloud resources to host and execute autonomic resources that will be used by services for their management such that the management cost is minimized while meeting the QoS requirements.

This chapter is organized as follows: Preliminaries on IaaS Cloud providers and deployed SBPs are firstly introduced in Section 5.2. Then we describe the problem of determining the best placement decisions of AMs for the management of SBPs in the Cloud, and we formulate it as an optimization model in the case where the number of AMs is known or not known in advance. We also present two different approaches for these two cases in Sections 5.4 and 5.5. Illustrative examples as well as experimental results are provided for both cases. Finally, we conclude the chapter in Section 5.6.

5.2 Preliminaries: IaaS Cloud provider, Deployed SBP

In this section, we present some definitions and basic concepts related to an IaaS Cloud provider and deployed SBPs.

5.2.1 IaaS Cloud Provider

An IaaS Cloud provider is hosted in multiple geographically distributed regions. Each region has multiple Availability Zones (AZs), i.e. multiple data centers in simple terms, and each one may offer different VM types. Furthermore, multiple instances of each VM type can be offered by a zone. Figure 5.1 depicts a simplified representation a Cloud infrastructure provider.

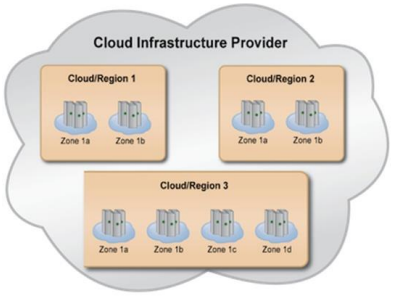


Figure 5.1: Cloud infrastructure provider [9].

Definition 5.2.1 (IaaS Cloud provider). *An IaaS Cloud provider is defined as a 3-tuple* $\langle \mathbb{Z}, cinter, cintra \rangle$ *which is inspired from a real Cloud provider*[1] *where:*

- $\mathbb{Z}$ *is the set of AZs. Each AZ offers several types of VMs (*$\mathbb{V}_z = \{vm\}$*),* $vm = (cu, ram, cpu, bw, avail, v_{max})$ *such that* cu *is the compute price* $(\$/h)$*,* ram *is the RAM capacity* (GB)*,* cpu *is the quantity of CPU cores,* bw *is the bandwidth capacity* (Mb/s)*,* $avail$ *is the availability capacity* $(\%)$*, and* v_{max} *is the maximum number of instances that can be deployed;*

- $cinter$ *is the inter-AZ data transfer price* $(\$/GB)$*;*

- $cintra$ *is the intra-AZ data transfer price* $(\$/GB)$*.*

5.2.2 Deployed SBP

A deployed SBP in the Cloud is represented by a graph, where services are deployed on different instances of VMs into different AZs.

Definition 5.2.2 (Deployed SBP). *A deployed SBP is defined as a 6-tuple* $\langle S, E, I, l, \varepsilon, \delta \rangle$ *where:*

- S *is the set of services;*

- $E \subseteq S \times S$ *is the set of edges that represents the data dependencies between services, such that* $(s_i, s_j) \in E$ *if the output data of service* i *is required for the execution of service* j*;*

- $I \subset S$ *is the set of initial services;*

- $l : S \rightarrow \mathbb{Z} \times \mathbb{V} \times \mathbb{I}$ *is a function that assigns, for each service* $s \in S$*, an instance* $i \in \mathbb{I}$ *of VM* $v \in \mathbb{V}_z$ *in AZ* $z \in \mathbb{Z}$*, where* s *is deployed.* $\mathbb{I} = \{1, ..., v_{max_{zv}}\}$*;*

- $\varepsilon : S \leftarrow \mathbb{N}$ *is a function that assigns, for each service* $s \in S$*, an estimated execution time;*

- $\delta : S \times S \leftarrow \mathbb{N}$ *is a function that assigns, for each pair of services* $(s_i, s_j) \in E$*, an estimated quantity of data to be transferred from* s_i *to* s_j*.*

[1] The IaaS Cloud is inspired from the Amazon EC2 https://aws.amazon.com/fr/ec2/pricing/on-demand/

5.3 Problem Description

Managing SBPs in the Cloud involves using autonomic resources to dynamically adapt services to changes. A key challenge faced by Cloud providers is to efficiently allocate Cloud resources to the services that make up these SBPs, which results in lower computing and communication costs while the required QoS is met. In fact, due to the diversity of services and autonomic resource QoS requirements as well as the heterogeneity of Cloud resources, and to ensure the performance of execution and other QoS aspects of SBPs, the computing resources allocated to them must satisfy the requirements of SBP services and autonomic resources that will be used by these services for their management in terms of hardware resources (CPU, RAM). Furthermore, these applications are characterized by their large data volume; which will result in a high amount of communication traffic between VMs. Consequently, it is also important to consider the placement decisions of services and autonomic resources in the Cloud. Since several research works (such as [44, 45, 49, 48]) have treated the problem of placement of services in the Cloud, in this chapter, we focus on the placement of autonomic resources that will be used by services that make up SBPs in the Cloud. The placement decisions of services will be considered as inputs.

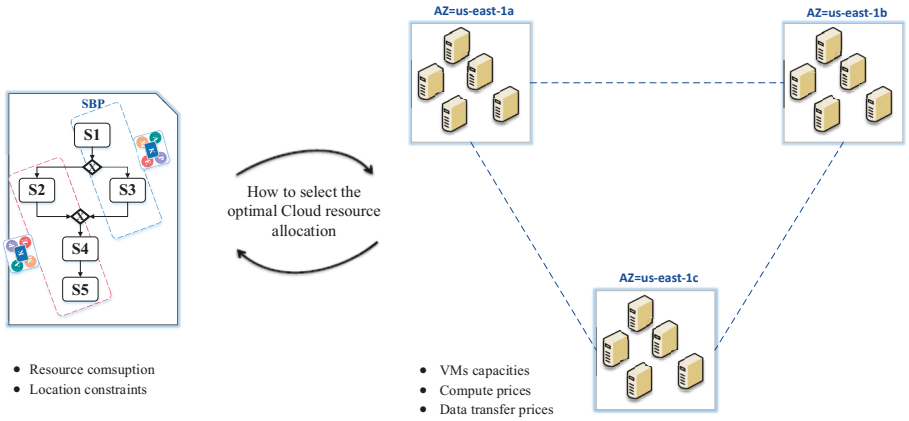


Figure 5.2: Second research problem: Finding the best placement decisions of AMs to be used by SBP services in the Cloud.

5.4 Placement of AMs in the Cloud for Efficient Management of SBPs

In this section, we present our first proposal for determining the best placement decisions of a given number of AMs that will be used by services that make up SBPs for their management in the Cloud (see the two previous chapters for more details). First, we

present an ILP formulation to solve this problem such that the total placement cost is minimized (Section 5.4.1). Afterwards, we present a near-optimal algorithm that provides good solutions in a polynomial time (Section 5.4.2). An illustrative example of our algorithm is then provided (Section 5.4.3). Afterwards, we present the experiments that we perform to evaluate it (Section 5.4.4).

5.4.1 Problem Formulation

We start this section by introducing some assumptions and notations in order to facilitate the formulation. Without loss of generality, we assume that all AMs have the same requirements in terms of hardware resources (RAM, CPU, etc.). Furthermore, we assume that all VMs are homogeneous, that is all VMs are characterized by the same hardware configuration in terms of CPU, RAM, etc. and have the same compute price. Given a deployed SBP, we denote by:

- S the set of services that make up SBP;
- PS the sets of services that can run in parallel;
- $\mathbb{Z}$ the set of AZs;
- $\mathbb{V}_z$ the set of VMs offered by AZ $z \in \mathbb{Z}$;
- M the set of AMs that will be used by services;
- D_v the set of services deployed on VM $v \in \mathbb{V}$;
- $data$ the estimated data transfer quantity between a service and an AM.

.

Towards this end, the following decision variables are defined:

- x_m^s takes 1 if AM $m \in M$ is assigned to service $s \in S$, and 0 otherwise;
- y_m^s takes 1 if AM $m \in M$ and service $s \in S$ are deployed on the same VM, and 0 otherwise;
- z_m^s takes 1 if AM $m \in M$ and service $s \in S$ are in the same AZ, and 0 otherwise;
- w_m^v takes 1 if AM $m \in M$ is deployed on VM $v \in V$, and 0 otherwise.

Given the above assumptions and notations, our problem can be formulated in equations [5.1-5.8] where the objective function 5.1 aims to minimize the sum of intra-AZ communication costs (the data transfer cost in the same VM is null) and the sum of inter-AZ communication costs. Constraint 5.2 makes sure that only one AM is assigned to each service all along the SBP life cycle. The authors in [75, 16] adopted the principle

of using multiple AMs for the management of applications to avoid management bottlenecks, but they did not provide any means to optimize their number. In our work, an AM is able to manage a set of sequential services to reduce the amount of monitoring data that will be processed by each AM to prevent bottlenecks. Therefore, constraint 5.3 ensures that parallel services are managed by different AMs. Constraint 5.4 guarantees that each AM is placed (deployed) on only one VM.

$$min \sum_{s \in S} \sum_{m \in M} cintra.data.x_m^s.z_m^s.(1 - y_m^s) + \sum_{s \in S} \sum_{m \in M} cinter.data.x_m^s.(1 - z_m^s) \quad (5.1)$$

Subject to:

$$\sum_{m \in M} x_m^s = 1 \qquad \forall s \in S \quad (5.2)$$

$$\sum_{s \in P} x_m^s <= 1 \qquad \forall m \in M,\ \forall P \in PS \quad (5.3)$$

$$\sum_{z \in Z} \sum_{v \in V_z} w_m^v = 1 \qquad \forall m \in M \quad (5.4)$$

Constraint 5.5 makes sure that δ_s^v takes 1 if service $s \in S$ is deployed on VM $v \in V$. We add equations [5.6-5.8] which ensure the linear relationships among the decision variables.

$$\delta_s^v = 1 \qquad \forall z \in Z,\ v \in V_z,\ s \in D_v \quad (5.5)$$

$$\sum_{v \in V_z} \delta_s^v + w_m^v - 2.z_m^s \leq 1 \qquad \forall z \in Z,\ \forall s \in S,\ \forall m \in M \quad (5.6)$$

$$\sum_{v \in V_z} \delta_s^v + w_m^v - 2.z_m^s \geq 0 \qquad \forall z \in Z,\ \forall s \in S,\ \forall m \in M \quad (5.7)$$

$$\delta_s^v + w_m^v - 2.y_m^s \geq 0 \qquad \forall z \in Z,\ \forall v \in V_z,\ \forall s \in S, \forall m \in M \quad (5.8)$$

Considering the number of deployments of applications in a public Cloud (see Chapter 3, Section 3.2.4 for more details), and as we will show in the evaluation section (Section 5.4.4), the time needed to solve the optimization model using the CPLEX solver is not acceptable. It can exceed two hours, and whenever the number of services goes beyond 21, the solver can not find the optimal solution. Consequently, finding the optimal solution, in this context, is not possible. Therefore, to tackle this NP-hard problem [80], we propose in the following section an algorithm that provides near-optimal solutions in a polynomial time.

5.4.2 Proposed Algorithm

In this section, we introduce our proposed algorithm, whose pseudo-code is given in Algorithm 5.1. It takes as inputs a deployed SBP and a number of AMs and outputs AM

assigned to each service, VM in which this AM is placed, and the total placement cost of AMs. It aims to determine the best placement decisions of these AMs in the Cloud such that the total transfer cost is minimized while fulfilling the constraints given by equations [5.1-5.8].

The algorithm starts by initializing the total placement cost to 0 (*cf.* line 1). After that, it starts with the first AM, and it looks for the set of maximum cardinality that contains the services that are deployed on the same VM and can run sequentially (*cf.* lines 6-10). The current AM is assigned to each service in this set, and it is placed in the current VM (*cf.* lines 11-14). This step is repeated until all the AMs are assigned to services (*cf.* line 15). Then for each service that does not have an AM assigned to it, the algorithm starts by AMs that are placed in the same AZ with this service in order to minimize the data transfer cost, and it checks whether there is an AM that can be assigned to it (*cf.* line 17). If it is the case, the latter AM is assigned to this service (*cf.* line 18). Otherwise, an AM that is placed into a different AZ is assigned to the current service (*cf.* line 21). The total placement cost is then updated (*cf.* line 19 and line 22).

Our Proposed algorithm consists in determining the best assignment decisions of a given number of AMs to the services that make an SBP as well as to VMs. The worst case time complexity of this algorithm is bounded by $O(n^2 \times m \times p)$, where n is the number of AMs, m is the number of VMs on which the services are deployed, and p is the number of sets of parallel services.

5.4.3 Illustrative Example

Herein, we present an example illustrating how the algorithm works. Let us consider the example depicted in Figure 5.3 that represents an example of an SBP composed of six services that are deployed on three homogeneous VMs.

In our example, we assume that the quantity of data to be transmitted from a service to an AM is 30. In addition, and for simplicity purposes, we assume that the sets of parallel services are: {S1}, {S2, S3}, {S2, S5}, {S4, S6}. The number of AMs that will be used by services is equal to 2.

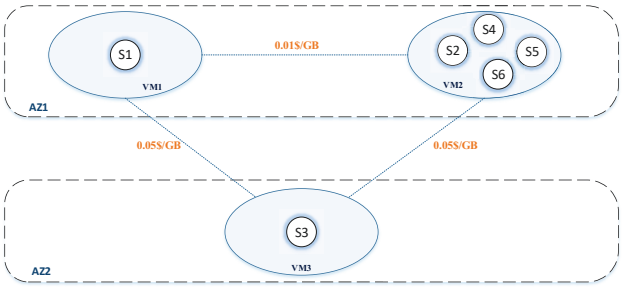


Figure 5.3: Example of SBP composed of six services deployed on three different homogeneous VMs.

Algorithm 5.1: AMsPlacement

Data: - $\langle S, E, I, l, \varepsilon, \delta \rangle$: Deployed SBP
- PS: Set of Sets of parallel services
- nbAMs: Number of AMs

Result: - AMs: Array containing the AM assigned to each service
- VMs: Array containing the VM on which each service is deployed
- totalcost: AMs placement cost

begin

1 $totalcost \leftarrow 0$;
2 $currentAM \leftarrow 0$;
3 **repeat**
4 $currentAM \leftarrow currentAM + 1$;
5 $maxCardinality \leftarrow 0$;
6 **for** $v_i \in V$ **do**
7 Find the set of sequential services with maximum cardinality G_i;
8 **if** $|G_i| > maxCardinality$ **then**
9 $maxCardinality \leftarrow |G_i|$;
10 $v_{min} \leftarrow v_i$;
11 **for** $s \in G_i$ **do**
12 $AMs[s] \leftarrow currentAM$;
13 $S \leftarrow S \setminus \{s\}$;
14 $VMs[currentAM] \leftarrow v_{min}$;
15 **until** $(currentAM = nbAMs)$;
16 **for** $s_i \in S$ **do**
17 **if** $\exists\ AM\ s.t.(VMs[AM]\ in\ AZ(s_i)$ ***and*** $\nexists\ s_j\ s.t.(AMs[s_j] = AM$ ***and*** $\{s_i, s_j\} \subseteq P \in PS))$ **then**
18 $AMs[s_i] \leftarrow AM$;
19 $totalcost \leftarrow totalcost + intra \times data$;
20 **else**
21 $AMs[s_i] \leftarrow \text{GetAM}(s_i)$;
22 $totalcost \leftarrow totalcost + inter \times data$;

First, the algorithm consists in grouping the services that are deployed on the same VM and can sequentially run in different groups to determine the set of sequential services with a maximum cardinality. In this example, it is $\{S2, S4\}$. As a consequence, AM number 1, *i.e.* $AM1$, is assigned to services $S2$ and $S4$, and $AM1$ is placed in VM2. The same process is applied to the rest of services, so AM number 2 is assigned to services $S5$ and $S6$, and $AM2$ is placed in VM2. The total placement cost is equal to 0 because the data transfer cost in the same VM is null. After that, the next iteration considers service $S1$. $AM1$ is assigned to $S1$ resulting in $cost = 0.01 \times 30 = 0.3\$$. For service $S3$, $AM1$ can not be assigned to it because it is already assigned to service $S2$ that can run in parallel with it, then $AM2$ is assigned to $S3$ resulting in $cost = cost + 0.05 \times 30 = 1.8\$$.

5.4.4 Experimental Results

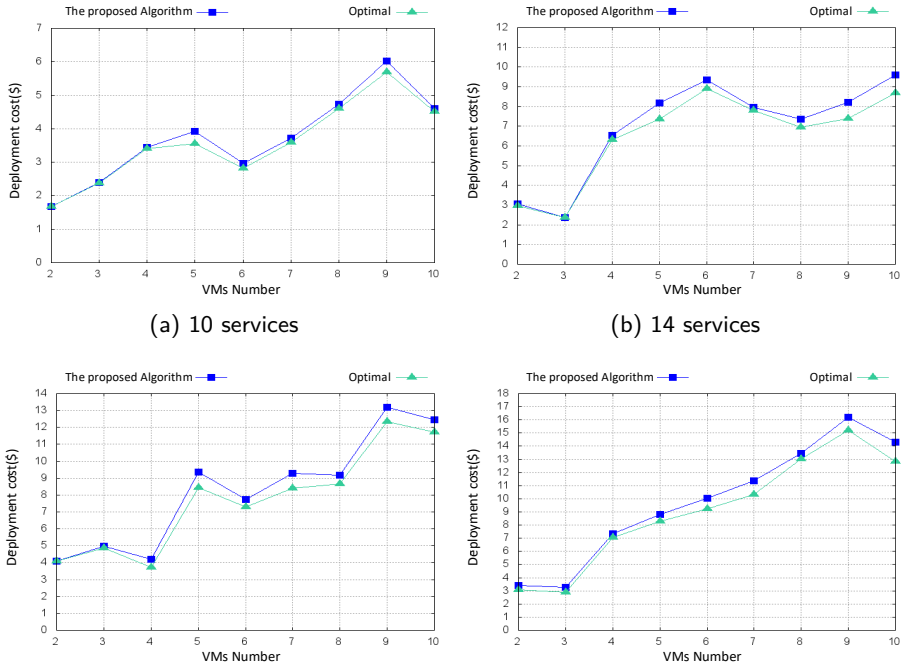
In this section, we present a series of performance evaluations of our algorithm in terms of execution time and quality of provided solutions. In order to show the efficiency of this approach, we compare it to the solution obtained by solving the formal model presented in Section 5.4.1 using the CPLEX solver, when such a solution is possible (see Chapter 3, Section 3.4.3 for more details).

To the best of our knowledge, none of the existing works has explicitly focused on the determination of the optimal placement decisions of AMs for the management of SBPs (see Chapter 2 for more details). Therefore, we do not make comparisons with existing work here. The different experiments are carried out on an Intel Core $i5$ PC with 2.53 GHz and 4GB of RAM. We use the commercial CPLEX solver 12.6 to solve the ILP formulation (Section 5.4.1).

In order to evaluate the performance of the proposed algorithm, we implement a generator of random SBPs (see the previous chapter, Section 4.6 for more details). For the experimental study, we vary the number of services from 2 to 22, in increments of 2. In addition, we vary the number of VMs form 2 to 10, in increments of 2. We consider that *cinter* is equal to $0.05\$$, *cintra* is equal to $0.01\$$, and the quantity of data exchange between a service and an AM is equal to 30.

The experimental results are depicted in Figure 5.4. We note that the gap between the objective function of our algorithm and the exact model is small and does not exceed 11.72%. It means that our solution is very close to the optimal solution.

In addition, the results show that our algorithm takes a very short time to provide the best placement decisions of AMs in the Cloud (less than 0.14 second). However, the exact method, solved by the ILP solver, takes more than two hours for 21 services, which are deployed on 10 different VMs. Note that the solver can not find the optimal solution even for small size problems (for more than 21 services) due to a problem of memory (Figure 5.5). In fact, the exact method is very time and resource-consuming due to the branch and bound algorithm that performs poorly.



(a) 10 services (b) 14 services

(c) 18 services (d) 21 services

Figure 5.4: First proposal: AMs placement cost versus number of services and VMs.

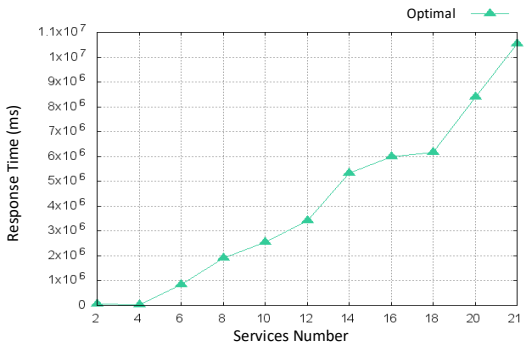


Figure 5.5: First proposal: Execution time of CPLEX versus number of services.

5.5 Joint Optimization of the Number of AMs and their Placement in the Cloud for Efficient Management of SBPs

In this section, we present our second proposal for determining the best placement decisions of AMs components for the management of SBPs, which will be determined during the placement process. First, we present an ILP formulation to solve this problem such that the total placement cost is minimized while meeting the QoS requirements (Section 5.5.1). Afterwards, we present a near-optimal approach that provides good solutions in a polynomial time (Section 5.5.2). An illustrative example of our approach is then provided (Section 5.5.3). Afterwards, we present the experiments performed to evaluate it (Section 5.5.4).

5.5.1 Problem Formulation

We start this section by introducing the notations that are used throughout the chapter as well as the assumptions. Next, we give the formulation of determining the best placement decisions of AM components that will be used by SBP services for their management in the Cloud.

Notations and Assumptions

Let S be the set of services that compose an SBP and PS the sets of parallel services. Let $\mathbb{Z}$ be the set of AZs, $\mathbb{V}_z$ be the set of VMs types offered by AZ $z \in \mathbb{Z}$, and $\mathbb{I}_{zv}$ be the set of instances of VM $v \in \mathbb{V}_z$. We define D_i as the set of services deployed on instance $i \in I$ of VM $v \in V$ into AZ $z \in Z$, and tet as the estimated execution time of SBP. Let M, A, P and E respectively represent the set of candidate monitors, analyzers, planners and executors that may be assigned to $|S|$. Each set has as cardinality the number of services, *i.e.* $|S|$, since in the worst case different monitors, analyzers, planners and executors will be assigned to the services. Let $rcpu_i$, $rram_i$ and $ravail_i$ respectively be the required quantity of CPU cores, RAM capacity and availability by AM component $i \in M \cup A \cup P \cup E$ or service $i \in S$.

We denote by t_{cv} the time taken by AM component c deployed on VM $v \in V$ to achieve its task, and by d_{kh} the quantity of data to be transmitted from service (respectively monitor, analyzer, planner, executor) k to monitor (respectively analyzer, planner, executor, service) h. Let mf_s be the monitoring frequency for service s and I_s the set of time stamps at which we can monitor it, where the step size between each element in I_s and the next element is equal to mf_s. We define Ev_s as the set of events that can result from the analysis of the monitoring data collected from service s. It contains the event 'no problem found' in addition to all the predefined problems that could happen. Let af_s be the analysis frequency for service s, and J_s be the set of time stamps at which we can analyze it, where the step size between each element in J_s and the next element is equal to af_s. We denote by p_{ev_s} the probability that problem ev will

happen for service s. Thus, $\forall s \in S, \sum_{ev_s \in Ev_s} p_{ev_s} = 1$. We also define Ac_s as the set of adaptation actions that can be produced by a planner component in order to process problems that could happen by service s. We denote by p_{ac_s} the probability that action ac will be chosen for service s. Hence, $\forall s \in S, \sum_{ac_s \in Ac_s} p_{ac_s} = 1$. Moreover, $alloc_{nt}$ takes 1 if instance $n \in \mathbb{I}$ of VM $v \in \mathbb{V}$ into AZ $z \in \mathbb{Z}$ is allocated to at least one service at time $t \in [0..tet]$, and 0 otherwise.

Without loss of generality, we assume that all monitors (respectively analyzers, planners, executors) have the same requirements in terms of hardware resources (RAM, CPU, etc.) and that resource requirements by services and AM components are static. We also assume that the time is divided into time slots of equal length $t \in [0..tet]$ and that each service i has a start time s_i and a termination time e_i.

Intervals

$C = \{1, 2, .., |S|\}$ is the set of services or the set of candidate monitors (respectively analyzers, planners, or executors).

$\mathbb{I} = \{1, .., v_{max_{zv}}\}$.

$\mathbb{T} = \{monitor, analyzer, planner, executor\}$.

Constants

$m = 1,\ a = 2,\ p = 3,\ e = 4\ and\ s = 5$ are respectively the identifiers of monitor, analyzer, planner, executor and service types.

Decision variables

In this formulation, we consider the following decision variables:

- x_{rjn} takes 1 if the candidate AM component $j \in C$ of type $r \in T$ is placed in instance $n \in \mathbb{I}$ of VM $v \in \mathbb{V}$ into AZ $z \in \mathbb{Z}$, and 0 otherwise.
- y_{kihj} takes 1 if the candidate AM component $i \in C$ of type $k \in T$ (or service $i \in C$) and the candidate AM component $j \in C$ of type $h \in T$ (or service $j \in C$) are in the same AZ, and 0 otherwise.
- z_{kihj} takes 1 if the candidate AM component $i \in C$ of type $k \in T$ (or service $i \in C$) and the candidate AM component $j \in C$ of type $h \in T$ (or service $j \in C$) are placed in the same VM, and 0 otherwise.
- w_{kjhi} takes 1 if the candidate AM component $j \in C$ of type $k \in T$ (or service $i \in C$) is linked/assigned to the candidate AM component $i \in C$ of type $h \in T$ (or service $i \in C$), and 0 otherwise.
- α_{ki} takes 1 if the candidate AM component $i \in C$ of type $k \in T$ is used by at least one AM component/service, and 0 otherwise.
- β_{sihjt} takes 1 if service $i \in C$ is assigned to the candidate AM component $j \in C$ of type $h \in T$ at time $t \in \{0, ..., tet\}$, and 0 otherwise.

- δ_{sihjv} takes 1 if service $i \in C$ is assigned to the candidate AM component $j \in C$ of type $h \in T$ and j is placed in VM $v \in \mathbb{V}$ into AZ $z \in \mathbb{Z}$, and 0 otherwise.
- γ_{nt} takes 1 if instance $n \in \mathbb{I}$ of VM $v \in \mathbb{V}$ into AZ $z \in \mathbb{Z}$ is allocated to at least one AM component at time $t \in \{0, ..., tet\}$, and 0 otherwise.

Constraints

The objective functions are subject to the following set of constraints:

(a) Resource constraints: The resource constraints impose that VM capacities in terms of RAM and CPU should satisfy the AM components and the services deployed on it:

$$\forall z \in \mathbb{Z},\ \forall v \in \mathbb{V}_z,\ n \in \mathbb{I}_{zv},\ \forall S \in PS,\ \forall t \in [0..tet]$$

$$\sum_{k \in S} rcpu_k.x_{skn} + \sum_{k \in S} \sum_{r \in T} \sum_{j \in C} rcpu_{kr}.x_{rjn}.\beta_{skrjt} \leqslant cpu_v \tag{5.1}$$

$$\sum_{k \in S} rram_k.x_{skn} + \sum_{k \in S} \sum_{r \in T} \sum_{j \in C} rram_{kr}.x_{rjn}.\beta_{skrjt} \leqslant ram_v \tag{5.2}$$

(b) QoS constraint: The QoS constraint imposes the minimum availability level required by an AM component:

$$avail_i.x_{rjn} \geq ravail_r.x_{rjn} \qquad \forall z \in \mathbb{Z}, \forall v \in \mathbb{V}_z, \forall n \in \mathbb{I}_{zv},\ \forall r \in T,\ \forall j \in C \tag{5.3}$$

(c) Placement constraints: The placement constraint (5.4) imposes that each AM component should be placed on one (and only one) VM. Constraint 5.5 ensures that x_{sjn} takes 1 if service $s \in S$ is placed in instance $n \in \mathbb{I}$ of VM $v \in \mathbb{V}$ into AZ $z \in \mathbb{Z}$.

$$\sum_{z \in \mathbb{Z}} \sum_{v \in \mathbb{V}_z} \sum_{n \in \mathbb{I}_{zv}} x_{rjn} = \alpha_{rj} \qquad \forall r \in T,\ \forall j \in C \tag{5.4}$$

$$x_{sjn} = 1 \qquad \forall z \in \mathbb{Z},\ \forall v \in \mathbb{V}_z,\ n \in \mathbb{I}_{zv},\ \forall j \in D_i \tag{5.5}$$

(d) Assignment constraints: Constraint 5.6 makes sure that only one monitor is assigned to each service all along the SBP life cycle. Constraint 5.10 ensures that if a monitor is used, then it is linked to only one analyzer. Constraint 5.14 guarantees that the latter analyzer is linked to only one planner. Constraint 5.18 ensures that the latter planner is linked to only one executor. Constraint 5.7 makes sure that if a monitor is assigned to at least one service, then it is considered as used. Constraint 5.11 (respectively 5.15, 5.19) makes sure that if an analyzer (respectively planner, executor) has at least one monitor (respectively analyzer, planner) linked to it, then it is considered as used. Constraints (5.8), (5.9), (5.12), (5.13), (5.16), (5.17), (5.20), and (5.21) guarantee that each component is assigned to at most one service at a time.

$\forall z \in \mathbb{Z},\ \forall v \in \mathbb{V}_z, \forall n \in \mathbb{I}_{zv},\ \forall j,k,h,l,o \in C$

$$\sum_{k \in C} w_{shmk} = 1 \tag{5.6}$$

$$w_{shmk} \leqslant \alpha_{mk} \tag{5.7}$$

$$w_{shmk}.x_{mkn} \leqslant \beta_{shmkt} \qquad \forall j \in I_h, \forall t \in \{j,..,j+t_{mv}-1\} \tag{5.8}$$

$$\sum_{h \in C} \beta_{shmkt} \leqslant 1 \qquad \forall t \in [0..tet] \tag{5.9}$$

$$\sum_{k \in C} w_{mhak} = \alpha_{mh} \tag{5.10}$$

$$w_{mhak} \leqslant \alpha_{ak} \tag{5.11}$$

$$\sum_{ev_l \in Ev_l} p_{ev_l}.w_{slmh}.w_{mhak}.x_{akn} \leqslant \beta_{slakt} \qquad \forall j \in J_l, \forall t \in \{j,..,j+t_{av}-1\} \tag{5.12}$$

$$\sum_{j \in C} \beta_{sjakt} \leqslant 1 \qquad \forall t \in [0..tet] \tag{5.13}$$

$$\sum_{k \in C} w_{ahpk} = \alpha_{ah} \tag{5.14}$$

$$w_{ahpk} \leqslant \alpha_{pk} \tag{5.15}$$

$$\sum_{ac_j \in Ac_j} p_{ac_j}.w_{sjmo}.w_{moah}.w_{ahpk}.x_{pkn} \leqslant \beta_{sjpkt} \qquad \forall t \in [s_j..e_j] \tag{5.16}$$

$$\sum_{j \in C} \beta_{sjpkt} \leqslant 1 \qquad \forall t \in [0..tet] \tag{5.17}$$

$$\sum_{k \in C} w_{phek} = \alpha_{ph} \tag{5.18}$$

$$w_{phek} \leqslant \alpha_{ek} \tag{5.19}$$

$$\sum_{ac_j \in Ac_j} p_{ac_j}.w_{sjmo}.w_{moal}.w_{alph}.w_{phek}.x_{ekn} \leqslant \beta_{sjekt} \qquad \forall t \in [s_j..e_j] \tag{5.20}$$

$$\sum_{j \in C} \beta_{sjekt} \leqslant 1 \qquad \forall t \in t \in [0..tet] \tag{5.21}$$

(e) Linearity constraints: We add constraints (5.22-5.38) which ensure the linear relationships among the decision variables.

$\forall z \in \mathbb{Z},\ \forall v \in \mathbb{V}_z,\ \forall n \in \mathbb{I}_{zv},\ \forall i,j,k,h,o \in C$

$$\delta_{simjv} = w_{simk}.x_{mkn} \tag{5.22}$$

$$\delta_{siajv} = w_{simk}.w_{mkaj}.x_{ajn} \tag{5.23}$$

$$\delta_{sipjv} = w_{simk}.w_{mkah}.w_{ahpj}.x_{pjn} \tag{5.24}$$

$$\delta_{siejv} = w_{simk}.w_{mkah}.w_{ahpo}.w_{poej}.x_{ejn} \tag{5.25}$$

We replace equation (5.22) in constraint (5.8), equation (5.23) in constraint (5.12), equation (5.24) in constraint (5.16), and equation (5.25) in constraint (5.20). Thus, constraints (5.8), (5.12), (5.16) and (5.20) respectively become:

$\forall k \in C, \forall h \in C$

$$\delta_{shmkv} \leqslant \beta_{shmkt} \qquad \forall j \in I_h,\ \forall t \in \{j,..,j+t_{mv}-1\} \tag{5.26}$$

$$\sum_{ev_h \in Ev_h} p_{ev_h}.\delta_{shakv} \leqslant \beta_{shakt} \qquad \forall j \in J_h, \forall t \in \{j,..,j+t_{av}-1\} \tag{5.27}$$

$$\sum_{ev_h \in Ev_h} p_{ev_h}.\delta_{shpkv} \leqslant \beta_{shpkt} \qquad \forall t \in [s_h..e_h] \tag{5.28}$$

$$\sum_{ev_h \in Ev_h} p_{ev_h}.\delta_{shekv} \leqslant \beta_{shekt} \qquad \forall t \in [s_h..e_h] \tag{5.29}$$

Then we must add the following logical constraints:

$\forall z \in \mathbb{Z},\ \forall v \in \mathbb{V}_z,\ \forall n \in \mathbb{I}_{zv},\ \forall i,j,k,h,o \in C$

$$w_{simk} + x_{mkn} \leq \delta_{simkv} + 1 \tag{5.30}$$

$$w_{simh} + w_{mhak} + x_{akn} \leq \delta_{siakv} + 2 \tag{5.31}$$

$$w_{simj} + w_{mjah} + w_{ahpk} + x_{pkn} \leq \delta_{sipkv} + 3 \tag{5.32}$$

$$w_{simo} + w_{moaj} + w_{ajph} + w_{phek} + x_{ekn} \leq \delta_{siekv} + 4 \tag{5.33}$$

$$w_{simk} + w_{mkah} + w_{ahpo} + w_{poej} \leq w_{ejsi} + 3 \tag{5.34}$$

$$\beta_{sihjt} + x_{hjn} \leq \gamma_{nt} + 1 \qquad \forall h \in T \tag{5.35}$$

$\forall z \in \mathbb{Z},\ \forall v \in \mathbb{V}_z,\ \forall n \in \mathbb{I}_{zv},\ \forall k,h \in T \cup \{s\}$

$$x_{kizvn} + x_{hjzvn} \leq z_{kihj} + 1 \tag{5.36}$$

$$\sum_{v \in \mathbb{V}_z} \sum_{n \in \mathbb{I}} x_{kin} + x_{hjn} - 2y_{kihj} \geq 0 \tag{5.37}$$

$$\sum_{v \in \mathbb{V}_z} \sum_{n \in \mathbb{I}} x_{kin} + x_{hjn} - 2y_{kihj} \leq 1 \tag{5.38}$$

Cost objective function: It consists in finding the minimum deployment cost of AM components that will be used to manage the given SBP, which includes: (i) the sum of VM allocation costs in order to execute these components, (ii) the sum of intra-AZ communication costs (the data transfer cost in the same VM is null), and (iii) the sum of inter-AZ communication costs.

$$\textbf{Min } cost = \sum_{z\in\mathbb{Z}}\sum_{v\in\mathbb{V}_z}\sum_{n\in\mathbb{I}_{zv}}\sum_{t=0}^{tet} cu_v.\gamma_{nt}.(1-alloc_{nt}) + \sum_{k\in T\cup\{s\}}\sum_{i\in C}\sum_{h\in T\cup\{s\}}\sum_{j\in C} cintra.$$
$$d_{kihj}.w_{kihj}.y_{kihj}.(1-z_{kihi}) + \sum_{k\in T\cup\{s\}}\sum_{i\in C}\sum_{h\in T\cup\{s\}}\sum_{j\in C} cinter.d_{kihj}.w_{kihj}.(1-y_{kihj})$$

We then retrieve sol^* the best solution found for *cost*, associated with the value $cost^* = cost(sol^*)$. Therefore, $cost^* = cost(sol^*)$ (5.39) becomes a constraint of the problem associated with *componentsNumber*.

ComponentsNumber objective function: It consists in finding the minimum number of AMs components (*componentsNumber* objective function and constraints [5.1-5.39]) while ensuring the minimum deployment cost (*cost* objective function and constraints [5.1-5.38]).

$$\textbf{Min } componentsNumber = \sum_{k\in T}\sum_{i\in C}\alpha_{ki}$$

Considering the number of deployments of applications in a public Cloud (see Chapter 3, Section 3.2.4 for more details), and as we will show in the evaluation section (Section 5.5.4), the time needed to solve the optimization model using the CPLEX solver is not acceptable. It can exceed two hours and whenever the number of services goes beyond 14, the solver can not find the optimal solution. Consequently, finding the optimal solution, in this context, is not possible. Therefore, to tackle this NP-hard problem [80], we propose in the following section an approach that provides near-optimal solutions in a polynomial time.

5.5.2 Proposed Approach

In this section, we introduce our proposed approach that operates in two steps. Step 1 is implemented using the AMsComponentsPlacement algorithm, whose pseudo-code is given in Algorithm 5.2. It aims to determine the best placement decisions of monitors, analyzers, planners and executors, which will be used by services for their management in the Cloud such that the total placement cost of these components is minimized while meeting the QoS requirements. Step 2 is implemented using the AMsComponentsNumber algorithm, whose pseudo-code is illustrated in Algorithm 5.3. Its main target is to reduce the number of AMs components while ensuring the placement cost determined in step 1.

5.5.2.1 AMsComponentsPlacement algorithm

The AMsComponentsPlacement algorithm takes as inputs a deployed SBP graph, the IaaS capacity in the form of VMs, and services and components requirements in terms of VM capacities in RAM, CPU, and availability. It outputs the monitor, the analyzer, the planner, and the executor assigned to each SBP service, on which VMs as well as in which AZs these latter components are placed as well as their placement cost.

The algorithm starts by initializing the total placement cost to 0 (*cf.* line 1). After that, for each service, it dedicates an AM that is placed in the VM where this service is deployed in order to avoid the communication and computing costs (*cf.* lines 2-3). The list of services is sorted in a decreasing order according to the total amount of resources (CPU then RAM) required by the dedicated components (*cf.* line 4). Next, for each service, AMsComponentsPlacement checks if the capacities of VM (on which the current service is deployed) in terms of RAM and CPU satisfy the AM components (*cf.* lines 5-6). If it is not the case, then for each component assigned to it where this component is placed on VM in which the current service is deployed (*cf.* line 8), the list of candidate VMs which can host the current component is determined (*cf.* lines 9-10). For each candidate VM, the cost of the placement of the current AM in this VM is calculated which takes into account inter-AZ and intra-AZ communication costs as well as VMs compute costs (*cf.* lines 11-12). The component to be chosen is the one with the minimum cost, and it is moved to VM on which it executes in minimum cost (*cf.* lines 13-17). Then the total placement cost is updated (*cf.* line 18). These steps are repeated until the resource constraints on VM, on which the current service is deployed, is not violated (*cf.* line 6).

5.5.2.2 AMsComponentsNumber algorithm

AMsComponentsNumber starts by minimizing the number of executors, planners, analyzers and monitors, which will be used by the SBP services in order not to push any monitor (respectively analyzer, planner) to link to more than one analyzer (respectively planner, executor). To do so, it checks for each service whether there is an executor that is not used by any service during the execution period of this service, and the latter executor guarantees its requirements while it is placed on VM where the executor assigned to it is placed to ensure the placement cost provided by the AMsComponentsPlacement algorithm. If it is the case, this executor is assigned to the current service (*cf.* lines 1-3). After that, the algorithm goes through the sets of services, where the services belonging to a set are managed by the same executor, set by set, to ensure that each planner is linked to only one executor (*cf.* lines 4-5). For each set, it looks for minimizing the number of planners assigned to this set (*cf.* lines 6-9). In the same way, it determines the appropriate number of analyzers and monitors (*cf.* lines 10-11).

Algorithm 5.2: AMsComponentsPlacement

Data: - $\langle S, E, I, l, \varepsilon, \delta \rangle$: Deployed SBP
Result: - M, A, P, E: Sets of, respectively, $\langle s, m, az, vm \rangle$, $\langle s, a, az, vm \rangle$, $\langle s, p, az, vm \rangle$, and $\langle s, e, az, vm \rangle$
- totalcost: AMs components placement cost

begin

$totalcost \leftarrow 0$;
for $s \in S$ **do**
  Assign to s an AM that satify its requirement and place it in the VM where s is deployed;
Sort S in descending order according to the total amount of resources (CPU, RAM) required by the AM components;
for $s \in S$ **do**
  while CAPACITY*(vm(s)) is violated* **do**
    $mincost \leftarrow +\infty$;
    for $c \in C_s$ ***where*** $c \in$ *vm(s)* **do**
      $CandidateVMs \leftarrow \emptyset$;
      Record in CandidateVMs the VMs that have a free capacity $\geq$ cap(c) during the execution period of c;
      for $v \in CandidateVMs$ **do**
        $cost \leftarrow intra \times ic(c) \times az(v, v_c) - intra \times oc(c, v) \times az(v, v_c)$
        $+inter \times ic(c) \times (1 - az(v, v_c))$
        $+(inter - intra) \times oc(c, azvms(v_c)) \times (1 - az(v, v_c))$
        $-inter * oc(c, v) \times (1 - az(v, v_c))$
        $+(intra - inter) \times oc(c, azvms(v) \setminus \{v\}) \times (1 - az(v, v_c))$
        $+cu_v \times time(c, v)$;
        /*$ic(c)$ returns the sum of the inner data transfer quantity between component c and the other components linked to it as well as the service to which c is assigned, where these latter are deployed on the same VM with c*/
        /*$oc(c, V)$ returns the sum of the outer data transfer quantity between c and the set of VMs V*/
        /*$az(v, v_c)$ takes 1 if VMs v and v_c are into the same AZ, and 0 otherwise*/
        /*$azvms(v)$ returns the set of VMs offered by the AZ containing VM v*/
        /*$time(v, c)$ returns the sum of time units where the VM v is only used by c*/
        if $cost < mincost$ **then**
          $mincost \leftarrow cost$;
          $c_{min} \leftarrow c_s$;
          $v_{min} \leftarrow v$;
    $\langle s, c_{min}, az(vm(s)), vm(s) \rangle \leftarrow \langle s, c_{min}, az(v_{min}), v_{min} \rangle$;
    $totalcost \leftarrow totalcost + mincost$;

Algorithm 5.3: AMsComponentsNumber

Data: - M, A, P, E: Sets of, respectively, $\langle s, m, az, vm\rangle$, $\langle s, a, az, vm\rangle$, $\langle s, p, az, vm\rangle$, and $\langle s, e, az, vm\rangle$

Result: - M, A, P, E: Sets of, respectively, ⟨s,m,az,vm⟩, ⟨s,a,az,vm⟩, ⟨ s,p,az,vm⟩, and ⟨s,e,az,vm⟩

begin

1 **for** $s \in S$ **do**

2 **if** $\exists\ e$ (requiredResources(e)$\geq$ requiredResources(e_s) ***and*** vm(e)=vm(e_s) ***and*** isNotUsed(e,s)) **then**

3 $\langle s, e_s, az, vm\rangle \leftarrow \langle s, e, az, vm\rangle$; /*update*/

4 Group the services into different sets, *i.e.* ESets;

5 **for** $set \in ESets$ **do**

6 **if** $|set| > 1$ **then**

7 **for** $s\ in\ set$ **do**

8 **if** $\exists p$ (requiredResources(p)$\geq$ requiredResources(p_s) ***and*** vm(p)=vm(p_s) ***and*** isNotUsed(p,s) ***and*** $p \in$ planner(set)) **then**

9 $\langle s, p_s, az, vm\rangle \leftarrow \langle s, p, az, vm\rangle$; /*update*/

10 Group the services into different sets, *i.e.* PSets, then look to minimize the number of analyzers;

11 Group the services into different sets, *i.e.* ASets, then look to minimize the number of monitors;

5.5.2.3 Theoretical Complexity

Our proposed approach consists of two algorithms. The first one aims to find the best placement decisions of AMs components for the management of an SBP (Algorithm 5.2). Its worst case time complexity is bounded by $O(n^2 * m * p)$, where n is the number of services that compose SBP, m is the number of VMs, and p is the maximum time from the estimated execution times of the SBP services. The second algorithm aims to minimize the number of AMs components while ensuring the total placement cost provided by Algorithm 5.2 (Algorithm 5.3). The worst case time complexity of this algorithm is bounded by $O(n^2 * p)$, where n is the number of services and p is the maximum time from the estimated execution times of the SBP services. In summary, we can say that the theoretical time complexity of our approach is bounded by $O(n^2 * m * p)$.

5.5.3 Illustrative Example

Herein, we present an example illustrating how the approach works. Let us consider the example depicted in Figure 5.6 that represents an example of an SBP composed of five services that are deployed on three heterogeneous VMs.

In our example, we assume that the resources required for each AM component to complete its execution are 200MB of RAM and one CPU. In addition, we assume that each service requires 500 MB of RAM and services $S1$, $S2$, $S3$, $S4$ and $S5$ respectively require 2, 4, 2, 2 and 4 CPUs to complete their execution. Furthermore, we assume that the quantity of data to be transmitted from service (respectively monitor, analyzer, planner, executor) to monitor (respectively analyzer, planner, executor, service) is 30 (respectively 25, 20, 25, 18). For simplicity purposes, we also assume that the sets of parallel services are: {S1, S2, S3}, {S4, S6}.

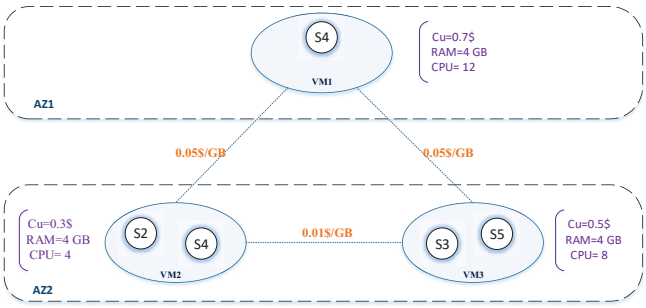


Figure 5.6: Example of SBP composed of five services deployed on three different heterogeneous VMs.

Initially, AMsComponentsPlacement assigns to each service an AM (monitor, analyzer, planner, and executor) which is placed in the same VM with it. Next, the services are sorted in a descending order according to the total amount of resources (CPU, RAM) required by their AMs: $S2$, $S5$, $S1$, $S3$, $S4$.

The first iteration of the while loop considers service $S2$. The capacity of $VM2$ is violated. The component to be moved is the one with the minimum cost. In this case, monitor $m2$ is moved from $VM2$ to $VM3$ resulting in cost=$0.01 \times (30+25) \times 1 = 0.55\$$ (intra $\times$ ic(m2) $\times$ az(VM3,VM2)). Then analyzer $a2$ is moved from $VM2$ to $VM3$ resulting in cost=$0.55 + 0.01 \times 20 \times 1 - 0.01 \times 25 \times 1 = 0.5\$$ (cost + intra $\times$ ic(a2) $\times$ az(VM3,VM2) - intra $\times$ oc(a2,VM3) $\times$ az(VM3,VM2)). $VM3$ can not host component $p2$ since $VM3$ has 8 CPUs which are used by $S3$, the AM components assigned to $S3$ and components $m2$ and $a2$ $(2+4+1+1)$. Thus, $p2$ is moved from $VM2$ to $VM1$ resulting in cost=$0.5 + 0.05 \times 25 \times 1 + 0.04 \times 20 \times 1 = 2.55\$$ (cost+ inter $\times$ ic(p2) $\times$ (1-az(VM1,VM2)+(inter-intra) $\times$ oc(p2, VM3) $\times$(1-az(VM1, VM2)). $e2$ is also moved from $VM2$ to $VM1$ resulting in cost=$2.55 + 0.05 \times 18 \times 1 - 0.05 \times 25 \times 1 = 2.2\$$ (cost+ inter $\times$ ic(e2) $\times$ (1-az(VM1,VM2)-inter $\times$ oc(e2, VM2) $\times$ (1-az(VM1, VM2)).

The second iteration of the while loop considers service $S5$. The capacity of $VM3$ satisfies the requirements of the components assigned to $S5$, so there are no component to move. The same process is applied to services $S1$ and $S3$.

The fifth iteration considers service $S4$. The capacity of $VM2$ is violated. Monitor $m4$ is moved from $VM2$ to $VM1$ resulting in cost=$2.2 + 0.05 \times (30+25) \times 1 = 4.95\$$ (inter $\times$ ic(m4) $\times$ az(VM1,VM2)). Then analyzer $a4$ is moved from $VM2$ to $VM1$ resulting in cost=$4.95 - 0.05 \times 25 \times 1 + 0.05 \times 20 \times 1 = 4.7\$$ (cost - inter $\times$ oc(a4,VM1) $\times$ az(VM1,VM2) + inter $\times$ ic(a4) $\times$ (1-az(VM1,VM2)).

5.5.4 Experimental Results

In this section, we present a series of performance evaluations of our approach in terms of execution time and quality of the provided solutions. In order to show the efficiency of this approach, we compare it to the solution obtained by solving the formal model presented in Section 5.5.1 using the CPLEX solver, when such a solution is possible. The different experiments are carried out on an Intel Core $i7$ PC with 2.70 GHz and 8GB of RAM. We use the commercial CPLEX solver 12.6 to solve the ILP formulation (Section 5.5.1).

In order to evaluate the performance of the proposed approach, we implement a generator of random SBPs (see the previous chapter, Section 4.6 for more details). For the experimental study, we vary the number of services from 2 to 14, in increments of 1, and from 10 to 100, in increments of 10. In addition, we consider that VMs have different hardware capacities. In particular, the following capacities are considered for VMs where at least a service is deployed on each one: Each VM may host 100%, 75%, 50%, 25%, or 0% of the components assigned to the services deployed on it; i.e., CPU and RAM capacities of a VM depend on the RAM and CPU required by the components. All the results are average values across 10 independent runs. All data inputs are randomly generated and detailed in Table 5.1[2].

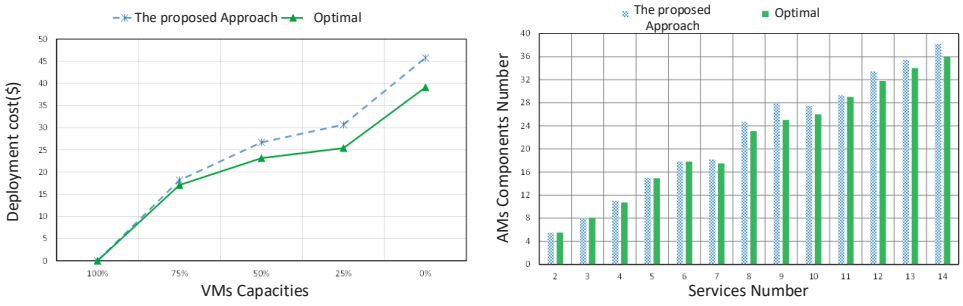
The proposed algorithm is able to find the optimal solution for more than 33% of

[2]The IaaS Cloud is inspired from the Amazon EC2 https://aws.amazon.com/fr/ec2/pricing/on-demand/

Information	Type	Range
Number of AZs ($\mathbb{Z}$)	Integer	[2..10]
Number of VM types ($\mathbb{V}$)	Integer	[1..86]
Number of maximum instances (v_{max})	Integer	[1..10]
inter-AZ communication cost ($cinter$)	Double	[0.01..0.10]
intra-AZ communication cost ($cintra$)	Double	[0.001..0.01]
Compute price (cu)	Double	[0.0058..5.424]
Availability level ($avail$)	Integer	[0.5..1]
Number of CPU cores (cpu)	Integer	[1..128]
RAM amount (ram)	Double	[0.5..1952]
Requirement in availability ($ravail$)	Double	[0.5..1]
Requirement in CPU ($rcpu$)	Integer	[1..32]
Requirement in RAM ($rram$)	Double	[0.5..488]

Table 5.1: Characteristics of data inputs of Cloud resources.

the considered SBPs. We note that the gap between the two solutions does not exceed 18.20%. It means that our solution is very close to the optimal solution (Figure 5.7). As for the execution time, the results presented in Figures 5.8 and 5.9 show that its execution time is reasonable. Indeed, it does not exceed 0.15 second, whereas CPLEX takes more than two hours to solve the problem, and whenever the number of SBP services goes beyond 14, the solver can not find the optimal solution due to a problem of memory.



(a) Total placement cost

(b) Number of components

Figure 5.7: Second proposal: Placement cost of AM components and their number versus number of services.

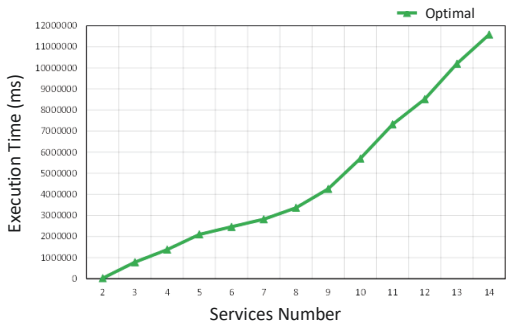


Figure 5.8: Second proposal: Execution time of CPLEX versus number of services.

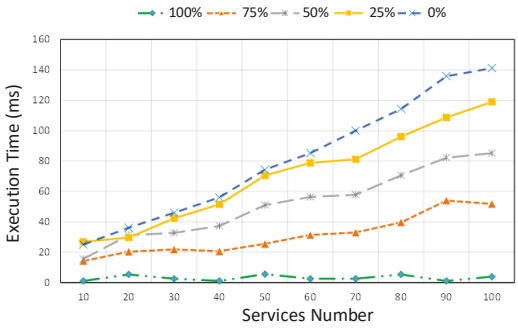


Figure 5.9: Second proposal: Execution time of proposed approach versus VMs capacities and number of Services.

5.6 Conclusion

In this chapter, we focused on the problem of the placement of autonomic resources for the management of timed SBPs in the Cloud. We first proposed an approach for determining the best placement decisions of a pre-determined number of AMs in the Cloud. Then we presented an approach that aims to combine the placement and determination of an appropriate number of AMs in the Cloud. An ILP formulation and an illustrative example of how the approach works were given for both cases. We presented the different experiments that we realized to show the efficiency of our proposals.

Chapter 6

Conclusion and Future Works

The research problem of this thesis is expressed by this interrogation: *How to determine the optimal Cloud resource allocation for the management of SBPs?* Previous chapters presented in details our solutions to answer this question. In this chapter, we summarize our contributions in Section 6.1 and present the future work in Section 6.2.

6.1 Contributions

Cloud Computing is an emerging paradigm in Information Technologies (IT). It refers to a model for the provision of every network-available resource "X" as a service "XaaS" based on the *pay-per-use* economic model. Cloud environments being increasingly used for hosting and executing applications that are described according to Service-Oriented Architecture (SOA) such as service-based business processes (SBPs) that we targeted in our work. Executing SBPs in dynamic environments requires autonomic management to cope with the dynamic evolution of Cloud environments with minimal human intervention. Autonomic management consists of a number of controlling devices known as Autonomic Managers (AMs).

Management of an SBP in Cloud environments has not received the needed attention in research work, particularly optimizing Cloud resource allocation for the management of SBPs that face many challenges : (1) allocating the adequate number of AMs to manage the process services and (2) allocating the required Cloud resource to host and execute these AMs.

In this regard, as a first contribution, we presented an approach that efficiently manages SBPs such that the number of used AMs is minimized while avoiding management bottlenecks. To do this, we modeled SBPs using directed graphs. We proposed a deterministic optimization model to solve this problem. We also suggested two main algorithms that provide a near-optimal solution for both SBPs that can be represented as graphs with and without cycles. An illustrative example of each algorithm is then provided. The dif-

ferent experiments that we performed on public real and randomly generated datasets of SBPs show the added value of our contribution in managing SBPs.

After facing the challenge of allocating the appropriate number of AMs for the management of SBPs, we were interested in the components that make up an AM (monitor, analyzer, planner and executor) to integrate these latter separately in the optimization process. We proposed a deterministic optimization model to find the optimal number of monitors, analyzers, planners and executors, which will be used by SBPs. We also suggested an approach that provides good solutions. An illustrative example as well as experimental results are provided and show the effectiveness of the proposed approach.

In order to go further in our reasoning, we proposed to provide solutions for the placement of AMs that will be used to manage SBPs in the Cloud. To this end, we proposed two approaches for the placement of AMs while considering two different contexts: Placement of a given number of AMs and placement of AMs that will be determined progressively during the placement process. These approaches consist in determining the optimal placement decisions of AMs in the Cloud such that the overall management cost is minimized while ensuring the required quality of service of SBPs. We used a deterministic optimization model for smaller graphs to benchmark our approaches that exhibit better scaling behavior.

6.2 Future work

In this thesis, we faced different complex problems related to optimal Cloud resource allocation for the management of SBPs. We solved many of them and we included others in our future work.

Our suggested approaches allow for the optimization of the number of AMs that will be assigned to the SBP services for their management. Nonetheless, decisions taken in isolation by an AM may indirectly interfere with the decision taken by other AMs and globally affect the performance of the whole SBPs. Hence, the coordination of AMs is a key task for the effective management of the whole business process. Several research works have, by and large, tackled the problem either in hierarchical fashion by adding AMs in charge of coordination or in centralized fashion by using a shared public knowledge for coordinating AMs which are assigned to services [35, 74, 8]. As a first extension of our work, we propose to extend our approaches to optimize not only AMs that will be explicitly used to manage the SBP services but also additional AMs that will be in charge of coordinating AMs in order to guarantee the effective management of the whole SBP.

The second extension consists in proposing a management tool based on an autonomic system for optimal deployment of SBPs in the Cloud integrating our proposed algorithms as well as other optimization algorithms regarding the deployment of the SBP services and the coordination of AMs. In fact, it would be interesting to be able to make optimal decisions for these problems when a new SBP deployment request is triggered. To do this, we can investigate the use of a higher-level AM. The latter is responsible for (1) collecting information relative to the interdependency relationship among the SBP services, the

assignment constraint, the resource consumption of different services and AMs, the Cloud resources capacities, etc., (2) analyzing the collected data, and (3) carrying out the assignment, placement and coordination decisions made by the planner component.

Evaluating the performance of allocation policies in real Cloud environments for different applications under critical conditions is a challenging task. Indeed, the use of a real Cloud environment is costly and makes the reproduction of some results an extremely difficult task due to the system size and configuration. Moreover, the evaluation of certain scenarios is not supported [83]. An alternative to that is the use of simulation tools that allow the reproduction of the tests and offer the possibility of evaluating the hypothesis and models under different conditions to cope with the possible performance degradation before deploying in a real Cloud. In addition, these simulation tools are free. In [84], the authors presented a study and comparison of existing Cloud simulation tools. As an extension of our work, we aim to utilize the Cloud simulator CloudSim [83] which is widely used in the literature [85, 46, 86] to allow the simulation of different Cloud and SBP configurations and come up with decisions regarding management resource amounts and their placement at low cost. To do this, we propose to implement an extension of the CloudSim simulator that enables the simulation of the autonomic management of SBPs. The extension will include an implementation of the dependency relationships between the SBP services and the implementation of our proposed algorithms.

Bibliography

[1] N. Serrano, G. Gallardo, and J. Hernantes, "Infrastructure as a service and cloud technologies," *IEEE Software*, vol. 32, no. 2, pp. 30–36, Mar 2015.

[2] IBM White Paper, "An architectural blueprint for autonomic computing," 2005.

[3] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin, *A Practical Guide to the IBM Autonomic Computing Toolkit*, 2004. [Online]. Available: http://books.google.fr/books?id=XHeoSgAACAAJ

[4] R. Buyya, R. Calheiros, and X. Li, "Autonomic Cloud computing: Open challenges and architectural elements," in *International Conference on Emerging Applications of Information Technology*, 2012.

[5] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, Oct 2004.

[6] R. Asadollahi, M. Salehie, and L. Tahvildari, "StarMX: A framework for developing self-managing Java-based systems," in *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, May 2009, pp. 58–67.

[7] C. Ruz, F. Baude, and B. Sauvan, "Component-based generic approach for reconfigurable management of component-based SOA applications," in *International Workshop on Monitoring, Adaptation and Beyond*, 2010.

[8] F. A. de Oliveira Jr., T. Ledoux, and R. Sharrock, "A framework for the coordination of multiple autonomic managers in cloud environments," in *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 2013.

[9] B. Adler, "Four steps to achieving high availability in the cloud," https://www.rightscale.com/blog/enterprise-cloud-strategies/four-steps-achieving-high-availability-cloud, 2012.

[10] P. M. Mell and T. Grance, "Sp 800-145. the nist definition of cloud computing," Gaithersburg, MD, United States, Tech. Rep., 2011. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf

[11] L. Wu, S. K. Garg, S. Versteeg, and R. Buyya, "Sla-based resource provisioning for hosted software-as-a-service applications in cloud computing environments," *IEEE Transactions on Services Computing*, vol. 7, no. 3, pp. 465–485, July 2014.

[12] I. Enterprise, *Cloud Computing Survey*, 2016. [Online]. Available: http://core0.staticworld.net/assets/2016/11/03/cloud_exec_summ_2016.pdf

[13] M. P. Papazoglou and W. J. v. d. Heuvel, "Blueprinting the cloud," *IEEE Internet Computing*, vol. 15, no. 6, pp. 74–79, Nov 2011.

[14] S. A. White, "Introduction to bpmn," Tech. Rep., 2006. [Online]. Available: http://www.omg.org/bpmn/Documents/IntroductiontoBPMN.pdf

[15] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language User Guide, The (2Nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[16] L. Florio and E. D. Nitto, "Gru: An approach to introduce decentralized autonomic behavior in microservices architectures," in *IEEE International Conference on Autonomic Computing (ICAC)*, July 2016, pp. 357–362.

[17] P. Hoenisch, C. Hochreiner, D. Schuller, S. Schulte, J. Mendling, and S. Dustdar, "Cost-efficient scheduling of elastic processes in hybrid clouds," in *IEEE International Conference on Cloud Computing*, June 2015, pp. 17–24.

[18] T. Mastelic, W. Fdhila, I. Brandic, and S. Rinderle-Ma, "Predicting resource allocation and costs for business processes in the cloud," in *IEEE World Congress on Services*, June 2015, pp. 47–54.

[19] M. Rekik, K. Boukadi, N. Assy, W. Gaaloul, and H. Ben-Abdallah, "A linear program for optimal configurable business processes deployment into cloud federation," in *IEEE International Conference on Services Computing (SCC)*, 2016, pp. 34–41.

[20] L. Hadded, F. B. Charrada, and S. Tata, "An efficient optimization algorithm of autonomic managers in service-based applications," in *International Conference on Cooperative Information Systems (CoopIS)*, 2015, pp. 19–37.

[21] L. Hadded, F. B. Charrada, and S. Tata, "Optimizing autonomic resources for the management of large service-based business processes (accepted for publication)," *IEEE Transactions on Services Computing*, 2018.

[22] L. Hadded, F. B. Charrada, and S. Tata, "Optimization and approximate placement of autonomic resources for the management of service-based applications in the cloud," in *International Conference on Cooperative Information Systems (CoopIS)*, 2016, pp. 175–192.

[23] L. Hadded, F. B. Charrada, and S. Tata, "Optimization of autonomic manager components in service-based applications," in *IEEE International Conference on Services Computing (SCC)*, 2017, pp. 370–377.

[24] L. Hadded, F. B. Charrada, and S. Tata, "Efficient resource allocation for autonomic service-based applications in the cloud," in *IEEE International Conference on Autonomic Computing (ICAC)*, 2018.

[25] N. Manohar, "A survey of virtualization techniques in cloud computing," in *International Conference on VLSI, Communication, Advanced Devices, Signals & Systems and Networking (VCASAN)*, India, 2013, pp. 461–470.

[26] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[27] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," in *International Conference on Web Information Systems Engineering (WISE)*, Dec 2003, pp. 3–12.

[28] W. M. Coalition, "Terminology and glossary (wfmctc-1011)," http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf, Tech. Rep., 1999.

[29] W. van der Aalst and A. ter Hofstede, "Yawl: yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245 – 275, 2005.

[30] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, Jul 2003.

[31] S. S. Rao, *Engineering Optimization: Theory and Practice: Fourth Edition*. John Wiley and Sons, 2009.

[32] D. P. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*. Prentice Hall International (UK) Ltd., 1994.

[33] R. Ribler, J. Vetter, H. Simitci, and D. Reed, "Autopilot: adaptive control of distributed applications," in *International Symposium on High Performance Distributed Computing*, Jul 1998, pp. 172–179.

[34] N. Belhaj, D. Belaïd, and H. Mukhtar, "Self-adaptive decision making for the management of component-based applications," in *International Conference on Cooperative Information Systems (CoopIS)*, 2017, pp. 570–588.

[35] O. Mola and M. A. Bauer, "Collaborative policy-based autonomic management: In a hierarchical model," in *International Conference on Network and Service Management*, Oct 2011, pp. 1–5.

[36] D. Ionescu, B. Solomon, M. Litoiu, and M. Mihaescu, "A Robust Autonomic Computing Architecture for Server Virtualization," in *International Conference on Intelligent Engineering Systems*, 2008.

[37] T. Baker, O. Rana, R. Calinescu, R. Tolosana-Calasanz, and J. Bañares, "Towards Autonomic Cloud Services Engineering via Intention Workflow Model," in *Economics of Grids, Clouds, Systems, and Services*, 2013.

[38] S. Frey, A. Diaconescu, D. Menga, and I. M. Demeure, "Towards a generic architecture and methodology for multi-goal, highly-distributed and dynamic autonomic systems," in *International Conference on Autonomic Computing (ICAC)*, 2013, pp. 201–212.

[39] D. Kurian and P. Chelliah, "An autonomic computing architecture for business applications," in *World Congress on Information and Communication Technologies*, 2012.

[40] S. M. K. Gueye, N. de Palma, and E. Rutten, "Component-based autonomic managers for coordination control," in *Coordination Models and Languages*, 2013, pp. 75–89.

[41] G. Martinovic and B. Zoric, "E-health Framework Based on Autonomic Cloud Computing," in *International Conference on Cloud and Green Computing*, 2012.

[42] M. Mohamed, M. Amziani, D. Belaïd, S. Tata, and T. Melliti, "An autonomic approach to manage elasticity of business processes in the cloud," *Future Generation Comp. Syst.*, vol. 50, pp. 49–61, 2015.

[43] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek, "Practical solutions for qos-based resource allocation," in *IEEE Real-Time Systems Symposium*, 1998, pp. 296–306.

[44] P. Hoenisch, D. Schuller, S. Schulte, C. Hochreiner, and S. Dustdar, "Optimization of complex elastic processes," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 700–713, Sept 2016.

[45] K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan, "Resources allocation and scheduling approaches for business process applications in cloud contexts," in

IEEE International Conference on Cloud Computing Technology and Science, Dec 2012, pp. 496–503.

[46] F. Fakhfakh, H. H. Kacem, and A. H. Kacem, "Dealing with structural changes on provisioning resources for deadline-constrained workflow," *The Journal of Supercomputing*, vol. 73, no. 7, pp. 2896–2918, 2017.

[47] X. Jiajie, L. Chengfei, Z. Xiaohui, and D. Zhiming, "Incorporating structural improvement into resource allocation for business process execution planning," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 3, pp. 427–442, 2012.

[48] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds," *Future Generation Comp. Syst.*, vol. 48, pp. 1–18, 2015.

[49] E. Goettelmann, W. Fdhila, and C. Godart, "Partitioning and cloud deployment of composite web services under security constraints," in *IEEE International Conference on Cloud Engineering (IC2E)*, March 2013, pp. 193–200.

[50] C. Labba, N. Assy, N. B. B. Saoud, and W. Gaaloul, "Adaptive deployment of service-based processes into cloud federations," in *Web Information Systems Engineering*, 2017, pp. 275–289.

[51] R. B. Halima, S. Kallel, W. Gaaloul, and M. Jmaiel, "Optimal cost for time-aware cloud resource allocation in business process," in *IEEE International Conference on Services Computing (SCC)*, June 2017, pp. 314–321.

[52] C. Jianfang, C. Junjie, and Z. Qingshan, "An optimized scheduling algorithm on a cloud workflow using a discrete particle swarm," *Cybern. Inf. Technol.*, vol. 14, no. 1, pp. 25–39, Mar. 2014.

[53] Z. I. M. Yusoh and M. Tang, "Composite saas placement and resource optimization in cloud computing using evolutionary algorithms," in *IEEE International Conference on Cloud Computing*, June 2012, pp. 590–597.

[54] B. Wu, C.-H. Chi, Z. Chen, M. Gu, and J. Sun, "Workflow-based resource allocation to optimize overall performance of composite services," *Future Generation Computer Systems*, vol. 25, no. 3, pp. 199 – 212, 2009.

[55] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158 – 169, 2013.

[56] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *IEEE International Conference on Advanced Information Networking and Applications*, April 2010, pp. 400–407.

[57] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[58] C. Ghribi and D. Zeghlache, "Exact and heuristic graph-coloring for energy efficient advance cloud resource reservation," in *IEEE International Conference on Cloud Computing*, Jun. 2014, pp. 112 – 119.

[59] K.-C. Huang and B.-J. Shen, "Service deployment strategies for efficient execution of composite saas applications on cloud platform," *Journal of Systems and Software*, vol. 107, pp. 127 – 141, 2015.

[60] Y.-T. Tsai, Y.-L. Lin, and F. Hsu, "The on-line first-fit algorithm for radio frequency assignment problems," *Information Processing Letters*, vol. 84, no. 4, pp. 195 – 199, 2002.

[61] H. Khalajzadeh, D. Yuan, J. Grundy, and Y. Yang, "Cost-effective social network data placement and replication using graph-partitioning," in *IEEE International Conference on Cognitive Computing (ICCC)*, June 2017, pp. 64–71.

[62] T. Verbelen, T. Stevens, F. D. Turck, and B. Dhoedt, "Graph partitioning algorithms for optimizing software deployment in mobile cloud computing," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 451 – 459, 2013.

[63] M. Tanaka and O. Tatebe, "Workflow scheduling to minimize data movement using multi-constraint graph partitioning," in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid)*, May 2012, pp. 65–72.

[64] S. Abrishami, M. Naghibzadeh, and D. H. J. Epema, "Cost-driven scheduling of grid workflows using partial critical paths," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1400–1414, Aug 2012.

[65] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[66] T. Yu and K.-J. Lin, "Service selection algorithms for web services with end-to-end qos constraints," in *IEEE International Conference on e-Commerce Technology*, July 2004, pp. 129–136.

[67] S. Mahmoudi and S. Lotfi, "Modified cuckoo optimization algorithm (mcoa) to solve graph coloring problem," *Applied Soft Computing*, vol. 33, pp. 48 – 64, 2015.

[68] R. Likaj, A. Shala, M. Mehmetaj, P. Hyseni, and X. Bajrami, "Application of graph theory to find optimal paths for the transportation problem," *IFAC Proceedings Volumes*, vol. 46, no. 8, pp. 235 – 240, 2013.

[69] Z. Sun, W. Guo, Z. Wang, Y. Jin, W. Sun, W. Hu, and C. Qiao, "Scheduling algorithm for workflow-based applications in optical grid," *Journal of Lightwave Technology*, vol. 26, no. 17, pp. 3011–3020, Sept 2008.

[70] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement," in *IEEE International Conference on Services Computing*, July 2011, pp. 72–79.

[71] L. Golab, M. Hadjieleftheriou, H. Karloff, and B. Saha, "Distributed data placement to minimize communication costs via graph partitioning," in *International Conference on Scientific and Statistical Database Management*, 2014, pp. 20:1–20:12.

[72] R. Dijkman, M. Dumas, and L. García-Bañuelos, "Graph matching algorithms for business process model similarity search," in *Business Process Management*, 2009, pp. 48–63.

[73] B. Kiepuszewski, A. H. M. ter Hofstede, and C. J. Bussler, "On structured workflow modelling," in *Advanced Information Systems Engineering*, 2000.

[74] N. Belhaj, I. B. Lahmar, M. Mohamed, and D. Belaïd, "Collaborative autonomic management of distributed component-based applications," in *International Conference on Cooperative Information Systems (CoopIS)*, 2015, pp. 3–18.

[75] V. Vlassov, A. Al-Shishtawy, P. Brand, and N. Parlavantzas, "Self-Management with Niche, a Platform for Self-Managing Distributed Applications," in *Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification*, 2011. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00519596

[76] S. Balakrishnan, "2.5m applications created, 1 billion requests per day, and more – new openshift online milestones," https://blog.openshift.com/2-5m-application-created-1-billion-requests-per-day-and-more-new-openshift-online-milestones/, 2015.

[77] IBM, "IBM ILOG CPLEX Optimization Studio," https://www.ibm.com/products/ilog-cplex-optimization-studio, visited: 24 January 2018.

[78] M. Reichert, "Visualizing large business process models: Challenges, techniques, applications," in *Business Process Management Workshops*, 2013, pp. 725–736.

[79] M. Hipp, A. Strauss, B. Michelberger, B. Mutschler, and M. Reichert, "Enabling a user-friendly visualization of business process models," in *Business Process Management Workshops*, 2015, pp. 395–407.

[80] S. Sahni, "Computationally related problems," *SIAM Journal on Computing*, vol. 3, no. 4, pp. 262–279, 1974.

[81] D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf, "Analysis on demand: Instantaneous soundness checking of industrial business process models," *Data & Knowledge Engineering*, vol. 70, no. 5, pp. 448 – 466, 2011.

[82] G. Keller and T. Teufel, *Sap R/3 Process Oriented Implementation*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[83] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *International Conference on High Performance Computing Simulation*, June 2009, pp. 1–11.

[84] K. Ettikyala and Y. R. Devi, "A study on cloud simulation tools," *International Journal of Computer Applications*, vol. 115, no. 14, pp. 18–21, April 2015.

[85] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurr. Comput. : Pract. Exper.*, vol. 24, no. 13, pp. 1397–1420, Sep. 2012.

[86] P. Humane and J. N. Varshapriya, "Simulation of cloud infrastructure using cloudsim simulator: A practical approach for researchers," in *International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, May 2015, pp. 207–211.