Optimal Autonomic Management of Service-Based Business Processes in the Cloud

Leila Hadded (hadded.leila@gmail.com)
Universite de Tunis El Manar Faculte des Sciences de Tunis https://orcid.org/0000-0003-1637-1167
Tarek Hamrouni
Universite de Tunis El Manar Faculte des Sciences de Tunis

Research Article

Keywords: Cloud computing, autonomic management, business process, deployment, linear program

Posted Date: January 4th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-749393/v1

Soft Computing manuscript No.
(will be inserted by the editor)

Optimal Autonomic Management of Service-Based Business Processes in the Cloud

Leila Hadded[1] · **Tarek Hamrouni**[1]

Received: date / Accepted: date

Abstract Cloud computing is an emerging paradigm that provides hardware, platform and software resources as services over the internet in a pay-as-you-go model. It is being increasingly used for hosting and executing service-based business processes. These business processes are exposed to dynamic evolution during their life-cycle due to the highly dynamic evolution of cloud environments. The main adopted technique is to couple cloud computing with autonomic management in order to build autonomic computing systems. Almost all the existing approaches on autonomic computing have been focused on modeling and implementing autonomic mechanisms without paying any attention to the optimization of the autonomic management cost. Therefore, in this paper, we propose a novel approach based on binary linear program for determining the optimal allocation of cloud resources to manage a service-based business process which guarantees the specific requirements of customers and minimizes the management monetary cost. Then, to validate our approach under realistic conditions and inputs, we extend the CloudSim simulator to model and simulate the behaviour of business processes and their management in a cloud environment. Experiments conducted on two real datasets highlight the effectiveness of our approach.

Keywords Cloud computing · autonomic management · business process · deployment · linear program

1 Introduction

Over the last decade, Cloud computing has appeared as a new model enabling on demand network access to shared configurable computing resources (*e.g.* networks,

Leila Hadded
E-mail: hadded.leila@gmail.com

Tarek Hamrouni
E-mail: tarek.hamrouni@fst.rnu.tn

[1]Computer Science Department, Faculty of Sciences of Tunis, University of Tunis El Manar, Tunis, Tunisia

servers, storage, applications, and services) which can be dynamically provisioned and released with minimal service provider interaction [28]. These resources are provided "as a service" over the Internet. The three main models of cloud services are: Infrastructure as a Service (IaaS), which provides computational resources in the form of Virtual Machines (VMs), Platform as a Service (PaaS), and Software as a Service (SaaS), which provides software applications and data. These applications are also known as service-based applications. Assembling services into an application can be ensured by using any appropriate service composition specification that can be either architecture-based (*e.g.* modeled in UML component diagram [6] or Service Component Architecture (SCA) [27]) or behavior-based (*e.g.* modeled in Business Process Execution Language (BPEL) [23]. In the latter case, applications are known as Service-based Business Processes (SBPs).

Autonomic computing has received great attention in the recent years, particularly in cloud computing, to automatically and dynamically adapt cloud resources and services to changing cloud environments in order to respond to the requirements of the business defined in Service Level Agreements (SLA) [22, 30]. The central component in an autonomic computing system is MAPE-K (Monitor, Analyzer, Planner, Executor, and Knowledge) loop, also known as Autonomic Manager (AM). An AM consists on periodically collecting monitoring data, analyzing them and generating reconfiguration actions to be executed on the managed system.

A single AM might not be sufficient to manage all services in an SBP, because each service may periodically generate a large data amount that needs to be processed by the AM. Nevertheless, using more AMs comes with some management cost. Autonomic management of SBPs is a new trend that faces several challenges among which how to find a cost-optimal allocation of cloud resources (*i.e.* AM, VM) to manage the SBP services while satisfying their Quality of Service (QoS) requirements. As a first preliminary attempt to tackle this issue, in our work presented in [17], we have proposed an approach for approximate placement of a predetermined number of AMs for managing applications in the cloud. However, this approach does not take into account the diversity of customers' requirements and the heterogeneity of resources offered by cloud providers. Then, we have proposed in [19] an algorithm for efficient resource allocation for autonomic applications in the cloud. In effect, the proposed approach [19] does not provide optimal solutions for the problem. To the best of our knowledge, the work we propose in this paper is the first to provide optimal solutions for autonomic management of SBPs in the cloud while considering QoS aspects and the heterogeneity of cloud resources.

In this paper, we consider a cloud scenario where a SaaS provider sells its autonomic resources to IaaS providers. In their turn, IaaS providers offer to their customers services (VMs, AMs) with QoS guarantees to host and run their SBPs subject to a set of QoS requirements. The major contributions of this paper are summarized as follows:

- Through studying the characteristics of the cloud and SBPs, we propose an exact optimization approach that selects the best VMs and AMs so as to achieve a minimum management cost of SBPs.
- We extend a popular and widely used cloud computing simulator (CloudSim) where we model autonmic SBPs. It provides the ability to model and simulate the execution and the autonomic management of SBPs in a cloud environment.

The remainder of this paper is structured as follows. Section 2 introduces the necessary background information in this work. Section 3 describes the proposed approach. Section 4 presents our experiments to validate and evaluate the performance and scalability of our proposal. Section 5 reviews the related works on autonomic computing in large scale environments. Section 6 concludes the paper and highlights future directions.

2 Background

In this section, we first introduce and define the key concepts used in this paper. We define the concept of autonomic management and we highlight the SBP definition. We then briefly describe the problem statement.

2.1 Autonomic Management

To achieve autonomic computing, IBM has proposed a reference model for autonomic controller [1] called autonomic manager, also known as the MAPE-K (Monitor, Analyzer, Planner, Executor, and Knowledge) loop.

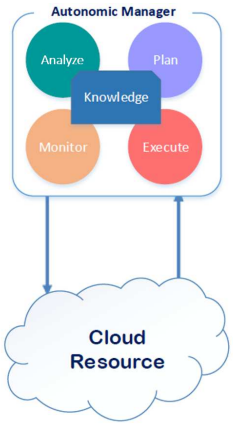

Fig. 1: Autonomic manager for a cloud resource.

In this autonomic loop, the central element represents any managed resource for which we want to exhibit an autonomic behavior. 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 to the planner;
3. The *Planner* is responsible for generating adaptation plans of actions to avoid violations of QoS;

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

2.2 SBP

An SBP is a set of related services that aim to accomplish a specific goal (see Fig. 2). A service is the smallest unit of work that offers computation or data capabilities. Assembling services into an SBP can be ensured using any appropriate service composition specification such as Event-driven Process Chain (EPC) [35] and Business Process Modeling Notation (BPMN) [15].

We formally define an SBP as a tuple ($\mathbb{S}$, $\mathbb{G}$, τ, $\mathbb{E}$) where:

- $\mathbb{S}$ is the non-empty set of services;
- $\mathbb{G}$ is the set of gateways;
- $\tau : \mathbb{G} \rightarrow \{AND, OR, XOR\}$ is a function that returns the type of each gateway. A gateway 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;
- $\mathbb{E}$ is the set of edges representing the control-flow of the process.

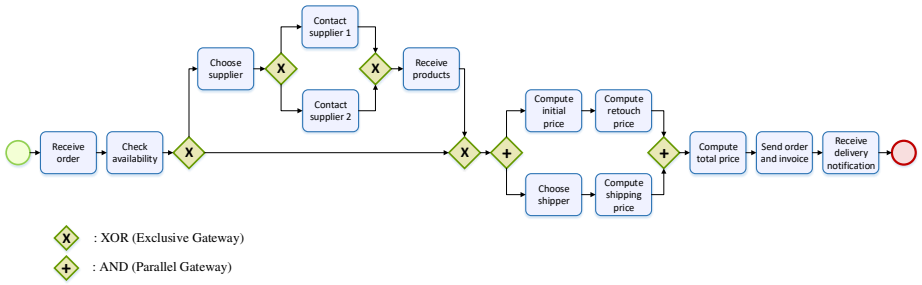

Fig. 2: Example of an SBP.

2.3 Problem statement

Autonomic management of a business process is typically realized using MAPE loops in order to respond to the requirements of the process based on SLAs.

IaaS cloud providers offer different types of VMs with different capabilities (*e.g.* memory, CPU power, bandwidth) and prices to fulfill customers' requirements. In addition, cloud providers, specifically SaaS providers, can offer a set of MAPE loops that have different resources requirements (*e.g.* memory, CPU power, bandwidth), QoS, and prices. In this paper, we consider a cloud scenario where a SaaS provider sells its autonomic resources to IaaS providers. In their turn, IaaS providers offer to their customers resources (VMs, AMs) with QoS guarantees to host and run their SBPs.

Since several research works have addressed the problem of optimal allocation of cloud resources to execute the SBP services (mapping between services and VMs) as in [2, 10, 37, 38], in this paper, we focus on how to determine a cost-optimal allocation of cloud resources (AMs, VMs) needed for the autonomic management of SBPs.

The problem we tackle in this paper is the following: Given a deployed SBP, our objective is to find the best mapping decisions between AMs and services as well as between AMs and VMs such that the management cost is minimized while meeting the QoS requirements. In the following section, we present our contribution to deal with this problem.

3 Proposed approach

In this section, we present our proposed approach for optimal autonomic management of SBPs in the cloud. Our objective is to find the optimal allocation of cloud resources (AMs, VMs) for the management of SBPs.

The customer requests the execution of its SBP subject to a set of QoS requirements that include the resource requirements (CPU, memory and bandwidth) and the minimum reliability and availability levels for executing the SBP services. Thus, the entry point of our proposal is a Unified Description Model (UDM) which describes SBP control-fow and QoS requirements. The UDM is defined as an XML document and it is composed of a set of nodes and edges. The nodes correspond to the services and the gateways of the SBP. The edges describe the interdependence between nodes. Listing 1 shows an extract of the UDM model of the SBP presented in Fig.2. Note that the IaaS cloud where the SBP will execute is abstracted from the customer.

```
<process>
<node type="startEvent" id="1"></node>
<node type="service" id="2" name="ReceiveOrder" size="312" rre="50" rav="80" rcpu="1" rram="
    25MB" rbw="120"></node>
<node type="service" id="3" name="CheckAvailability" size="250" rre="70" rav="70" rcpu="2"
    rram="30MB" rbw="150"></node>
<node type="exclusiveGateway" id="4"></node>
<node type="service" id="5" name="ChooseSupplier" size="300" rre="80" rav="70" rcpu="2" rram=
    "35MB" rbw="90"></node>
<node type="parallelGateway" id="6"></node>
<node type="service" id="7" name="ContactSupplier1" size="212" rre="50" rav="80" rcpu="1"
    rram="40MB" rbw="100"></node>
<node type="service" id="8" name="ContactSupplier2" size="212" rre="50" rav="80" rcpu="1"
    rram="40MB" rbw="100"></node>
<node type="parallelGateway" id="9"></node>
<node type="service" id="10" name="ReceiveProducts" size="300" rre="80" rav="80" rcpu="2"
    rram="32MB" rbw="150"></node>
<node type="exclusiveGateway" id="11"></node>
   ...
<node type="endEvent" id="21"></node>
<edge from="1" to="2"></edge>
<edge from="2" to="3"></edge>
<edge from="3" to="4"></edge>
<edge from="4" to="5"></edge>
<edge from="4" to="11"></edge>
<edge from="5" to="6"></edge>
<edge from="6" to="7"></edge>
<edge from="6" to="8"></edge>
<edge from="7" to="9"></edge>
<edge from="8" to="9"></edge>
<edge from="9" to="10"></edge>
```

```
<edge from="10" to="11"></edge>
    ...
</process>
```

Listing 1: Unified description model based on Fig.2.

The objective of a cloud provider is to execute the SBP with a minimum management cost while meeting all consumer requirements. This challenge is addressed through the following two components (see Fig. 3):

- Autonomic SBP Simulator: lies in extending Cloudsim simulator (see Fig. 4) with the modelling of (i) SBPs, and (ii) autonomic managers in order to simulate the behaviour of autonomic SBPs in a cloud environment. The objective is to estimate the management cost and the execution time needed for executing and managing SBPs with different cloud resource configurations;
- Autonomic SBP Optimizer: finds an optimal autonomic management of SBPs in the cloud.

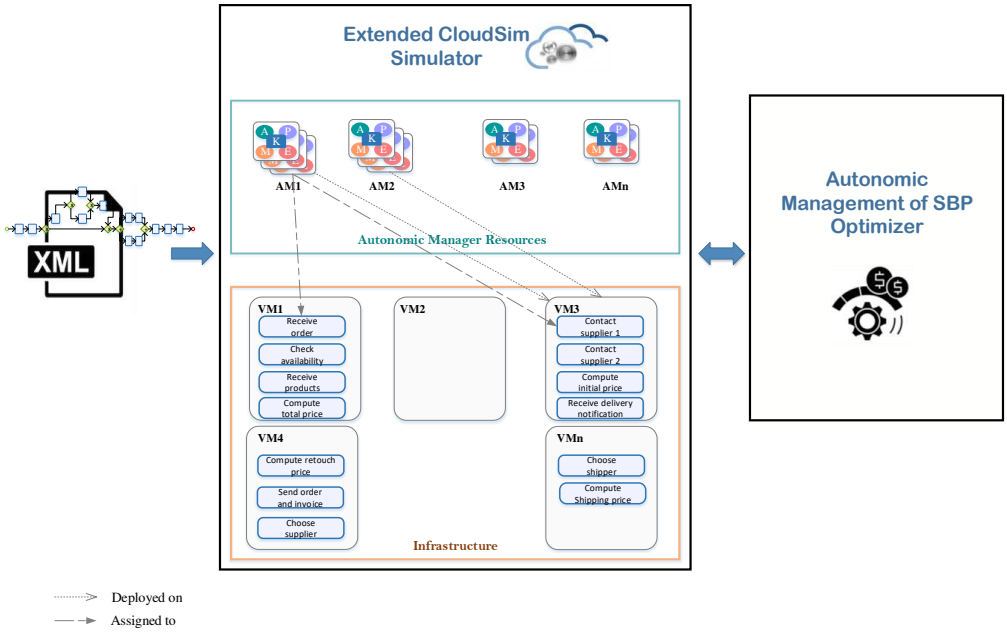

Fig. 3: Overview of the proposed approach.

3.1 Autonomic SBP simulator

Many simulation techniques to investigate the behavior of cloud computing have been developed [7, 8, 9, 25, 32]. One of the most widely used cloud simulators is

CloudSim, which is an open source, java-based simulator that enables seamless modeling, simulation, and experimentation of cloud computing environments. It supports more functionalities than other simulation tools and it is flexible. The Table 1 details a synthesis of cloud simulators.

Table 1: Summary of related cloud simulators.

Simulator	Programming Language	Resources		
		IaaS	PaaS	SaaS
CloudSim[8]	Java	Yes	Yes	No
GridSim[7]	Java	Yes	No	No
SimGrid[9]	C	Yes	No	No
GreenCloud[25]	C++,OTcl	Yes	No	No
iCanCloud[32]	C++	Yes	HPC	No

As Cloudsim is the best choice to simulate cloud computing resource [3], we design and implement an extension of CloudSim (the colored classes) as illustrated in Fig 4, where we can simulate the autonomic management of SBPs in cloud environments.

Before we present our extension, we first introduce the core components of Cloudsim:

- Datacenter: this class behaves like an infrastructure cloud providers (*e.g.* Amazon, Azure, App Engine). It models the main hardware resource that provides services for servicing customer requests;
- DatacenterBroker: this class represents a broker which acts as a mediator between customers and datacenters. It represents the customers needs;
- Host: is a physical resource (a computer) characterized by a number of CPU, memory, bandwidth, and storage capabilities;
- VirtualMachine: is a software-based emulation of a computer, which is managed and hosted by a host;
- RamProvisioner: this class represents the provisioning policy for allocating memory (RAM) to VMs that are deployed on a host;
- BwProvisioner: this class describes the provisioning policy of bandwidth to VMs that are deployed on a host;
- VmScheduler: this class models the policy required for allocating processor cores to VMs running in a host;
- Cloudlet: this class models an application component/service that run on a VM.

To model autonomic SBPs, the following classes have been designed:

- SBP: this class models the service-based business process to be executed on the cloud. An SBP consists of a set of services that are executed in a specific order according to gateways to achieve a specific business objective;
- Service: it is an extended class of *Cloudlet*. This class specifies the QoS requirements that include the resource requirements (CPU, memory and bandwidth) and the minimum reliability and availability levels for executing a service;
- Gateway: defines how an SBP behaves;

– AutonomicManager: this class represents an AM. An AM is a feedback control loop consisting of a monitor, an analyzer, a planner, and an executor, which share a knowledge base;
– Monitor, Analyzer, Planner, Executor: each one is an extended class of *Cloudlet*.

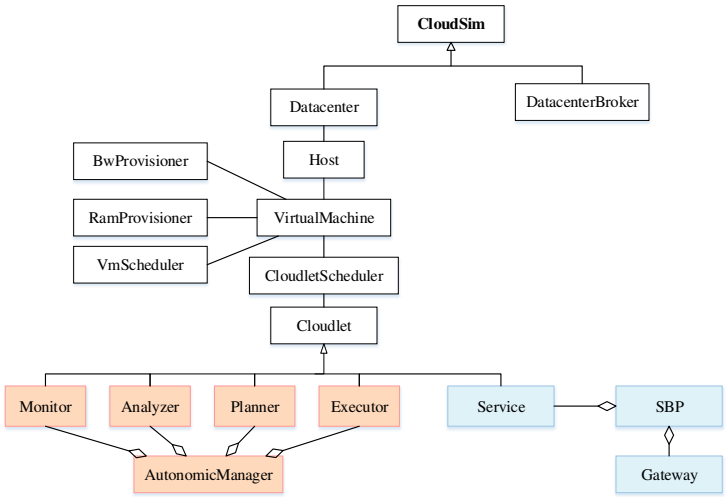

Fig. 4: Class design diagram of the main added elements in CloudSim.

3.2 Autonomic SBP optimizer

In order to find the optimal autonomic management of an SBP in the cloud, we propose an exact optimization model. The proposed Linear Program (LP) model is defined in terms of its decision variables, objective function, and constraints. It takes as inputs:

– An SBP is the tuple ($\mathbb{S}$, $\mathbb{G}$, τ, $\mathbb{E}$).
– A service is a tuple (rre, rav, dt, $rcpu$, $rram$, rbw, len), where rre and rav are, respectively, its minimum required reliability (%) and availability (%) level. dt is the size of data transferred from this service (MB), $rcpu$ is the minimum required CPU capacity (cores), $rram$ is the minimum required RAM capacity (GB), rbw is the minimum required bandwidth (MB/s), and len is the service's length/size in millions of instructions (MI).
– A VM is defined as a tuple (re, av, cp, dtp, cpu, ram, bw, $imax$, $mips$), where re and av are, respectively, the capability of the VM in terms of reliability (%) and availability (%). cp is the computed price ($/hour), dtp is the data transfer price ($/Mb), cpu is the CPU capacity (cores), ram is the RAM capacity (GB), bw is the bandwidth capacity (MB/s), $imax$ is the maximum number of instances of the VM, and $mips$ is the CPU speed in millions of instructions per second (MI/s). We denote by $\mathbb{I}_{vk} = \{1, 2, ..., imax_k\}$ the indexes of instances of the VM k.

- α denotes the maximum VM resource consumption (%) in order to avoid the overload of the VM.
- An AM is formalized as a tuple (rre, rav, mp, dt, $rcpu$, $rram$, rbw, $imax$, len), where rre and rav are, respectively, its minimum required reliability (%) and availability (%) level. mp is the AM price (\$/ hour), dt is the size of data transferred from the AM to a service (MB), $rcpu$ is the minimum required CPU capacity (cores), $rram$ is the minimum required RAM capacity (GB), rbw is the minimum required bandwidth (MB/s), $imax$ is the maximum number of instances of the AM, and len is its estimated size (MI). We denote by $\mathbb{I}_{mi} = \{1, 2, ..., imax_i\}$ the indexes of instances of the AM i.
- $dep(p, k, h) : \mathbb{S} \times \mathbb{V} \times \mathbb{I} \rightarrow \{0, 1\}$ is a location function that for each service p associates 1 if it is deployed in the instance h of the VM k, and 0 otherwise.

The execute time of an AM i on a VM k is calculated by dividing the size of the AM len_i by the CPU speed $mips_k$ multiplied by the number of cores of a CPU cpu_k, which can be formulated as:

$$et_{ki} = \frac{len_i}{mips_k \times cpu_k} \tag{1}$$

We assume that parallel services will not share any resources.

• Decision variables

We define the following decision variables:

- X_{ijkh} is equal to 1 if the instance $j \in \mathbb{I}_{mi}$ of the AM $i \in \mathbb{M}$ is deployed in the instance $h \in \mathbb{I}_{vk}$ of the VM $k \in \mathbb{V}$, and 0 otherwise;
- Y_{ijp} is equal to 1 if the instance $j \in \mathbb{I}_{mi}$ of the AM $i \in \mathbb{M}$ and the service $p \in \mathbb{S}$ are deployed in the same VM, and 0 otherwise;
- Z_{ijp} is equal to 1 if the instance $j \in \mathbb{I}_{mi}$ of the AM $i \in \mathbb{M}$ is assigned to the service $p \in \mathbb{S}$, and 0 otherwise.

• Cost objective function

The proposed objective function selects the AMs and VMs so as to achieve a minimum autonomic management cost of SBPs, including the total compute and communication costs:

(i) The compute cost is the sum of AM and VM allocation costs. Here, the allocation cost is the AM execution time et multiplied by the sum of the AM utilization price and the VM utilization price ($cp + mp$);

(ii) The communication cost is the sum of data transfer costs between AMs and services. The data transfer cost is equal to the bandwidth utilization price dtp multiplied by the transferred data size dt. The data transfer cost is deemed as negligible if the AM and the service are running on the same VM.

Hence, the objective function of our mathematical model takes the following form:

$$min \sum_{i=1}^{|\mathbb{M}|} \sum_{j=1}^{|\mathbb{I}_{mi}|} \sum_{k=1}^{|\mathbb{V}|} \sum_{h=1}^{|\mathbb{I}_{vk}|} et_{ki}(cp_k + mp_i) X_{ijkh} + \sum_{p=1}^{|\mathbb{S}|} \sum_{i=1}^{|\mathbb{M}|} \sum_{j=1}^{|\mathbb{I}_{mi}|} \sum_{k=1}^{|\mathbb{V}|} \sum_{h=1}^{|\mathbb{I}_{vk}|} dtp_k dt_{pi} X_{ijkh}(1 - Y_{ijp}) Z_{ijp} \tag{2}$$

• **Constraints**

The above objective function is subject to the following set of constraints:

– *QoS constraint:*

(i) VM constraints: ensure the minimum reliability (3) and availability (4) levels required by an AM to deploy on a VM.

$$\forall i \in \mathbb{M},\ \forall j \in \mathbb{I}_{mi},\ \forall k \in \mathbb{V},\ \forall h \in \mathbb{I}_{vk}$$

$$re_{vk} X_{ijkh} \geqslant rre_{mi} X_{ijkh} \tag{3}$$

$$av_{vk} X_{ijkh} \geqslant rav_{mi} X_{ijkh} \tag{4}$$

(ii) AM constraints: ensure the minimum reliability (5) and availability (6) levels required by a service to manage by an AM.

$$\forall i \in \mathbb{M},\ \forall j \in \mathbb{I}_{mi},\ \forall p \in \mathbb{S}$$

$$re_{mi} Z_{ijp} \geqslant rre_{sp} Z_{ijp} \tag{5}$$

$$av_{mi} Z_{ijp} \geqslant rav_{sp} Z_{ijp} \tag{6}$$

– *Resource constraints:* represent the VM's capacities constraints. These constraints ensure that the resources utilization of a VM is less than a threshold α. The term α forces the system to work away from the saturation point.

$$\forall i \in \mathbb{M},\ \forall j \in \mathbb{I}_{mi},\ \forall k \in \mathbb{V},\ \forall h \in \mathbb{I}_{vk},\ \forall p \in \mathbb{S}$$

$$X_{ijkh} rcpu_{mi} + X_{ijkh} Z_{ijp} Y_{ijp} rcpu_{sp} \leqslant \alpha cpu_k \tag{7}$$

$$X_{ijkh} rram_{mi} + X_{ijkh} Z_{ijp} Y_{ijp} rram_{sp} \leqslant \alpha ram_k \tag{8}$$

$$X_{ijkh} rbw_{mi} + X_{ijkh} Z_{ijp} Y_{ijp} rbw_{sp} \leqslant \alpha bw_k \tag{9}$$

– *Assignment constraint:* ensures that each service is managed by only one AM.

$$\sum_{i=1}^{|\mathbb{M}|} \sum_{j=1}^{\mathbb{I}_{mi}} Z_{ijp} = 1 \quad \forall p \in \mathbb{S} \tag{10}$$

– *Placement constraint:* ensures that each allocated AM is assigned to only one VM.

$$\forall i \in \mathbb{M},\ \forall j \in \mathbb{I}_{mi},\ \forall p \in \mathbb{S}$$

$$\sum_{k=1}^{|\mathbb{V}|} \sum_{h=1}^{\mathbb{I}_{vk}} X_{ijkh} \geqslant Z_{ijp} \tag{11}$$

– *Linearity constraints:* Non-linearity in the above constraints (7-9) is contributed by the product term $X_{ijkh} Z_{ijp} . Y_{ijp}$, where X_{ijkh}, Y_{ijp} and Z_{ijp} are binary decision variables. To linearize the above equations the following constraints are incorporated:

$$\forall i \in \mathbb{M},\ \forall j \in \mathbb{I}_{mi},\ \forall k \in \mathbb{V},\ \forall h \in \mathbb{I}_{vk},\ \forall p \in \mathbb{S}$$

$$W_{ijpkh} = X_{ijkh} Z_{ijp} Y_{ijp} \tag{12}$$

We replace the equation (12) in the constraints (7-9):

$\forall i \in \mathbb{M},\ \forall j \in \mathbb{I}_{mi},\ \forall k \in \mathbb{V},\ \forall h \in \mathbb{I}_{vk},\ \forall p \in \mathbb{S}$

$$X_{ijkh} rcpu_{mi} + W_{ijpkh} rcpu_{sp} \leqslant \alpha cpu_k \quad (13)$$

$$X_{ijkh} rram_{mi} + W_{ijpkh} rram_{sp} \leqslant \alpha ram_k \quad (14)$$

$$X_{ijkh} rbw_{mi} + W_{ijpkh} rbw_{sp} \leqslant \alpha bw_k \quad (15)$$

– *Logical constraints:* we add equations (16-19) to guarantee the relationships between the decision variables.

$\forall i \in \mathbb{M},\ \forall j \in \mathbb{I}_{mi},\ \forall k \in \mathbb{V},\ \forall h \in \mathbb{I}_{vk},\ \forall p \in \mathbb{S}$

$$W_{ijpkh} \geqslant X_{ijkh} + Z_{ijp} + Y_{ijp} - 2 \quad (16)$$

$$Y_{ijp} \geqslant X_{ijkh} + dep(p,k,h) - 1 \quad (17)$$

$$Y_{ijp} \leqslant X_{ijkh} \quad (18)$$

$$Y_{ijp} \leqslant dep(p,k,h) \quad (19)$$

– *Binary constraints:* constraints (20-23) ensure that the decision variables are either 0 or 1.

$\forall i \in \mathbb{M},\ \forall j \in \mathbb{I}_{mi},\ \forall k \in \mathbb{V},\ \forall h \in \mathbb{I}_{vk},\ \forall p \in \mathbb{S}$

$$X_{ijkh} \in \{0,1\} \quad (20)$$

$$Y_{ijp} \in \{0,1\} \quad (21)$$

$$Z_{ijp} \in \{0,1\} \quad (22)$$

$$W_{ijpkh} \in \{0,1\} \quad (23)$$

4 Validation and evaluation

In this section, we present experiments designed to evaluate the quality and the performance of our approach, measured in terms of management cost and response time as well as its flexibility measured in terms of the ability of the LP to cope with new constraints and new resource capacities.

Our approach is evaluated on two public real datasets of business process models from IBM [13] and the SAP reference model [24]. The first experiment compares our approach against our former work [17]. The second deals with adding a deployment constraint and scaling up and down the cloud resources capacities. However, To the best of our knowledge, there are no other existing research studies that focus on the problem addressed in this paper. Therefore, we are not able to cover other comparisons.

All experiments were carried out on a laptop equipped with an Intel® Core™ i7-4750HQ with 2.00 GHz processor and 12 GB of memory. The commercial solver CPLEX 12.6.3 [20] is used to solve the optimization problem.

4.1 Experimental parameters

In our experiments, we use real business process models from two large public datasets:

1. 560 BPMN process models [23] shared by the IBM Business Integration Technologies (BIT) team [13]. The number of services in these SBPs varies between 2 and 69.
2. 205 process model of SAP reference models [24] represented in EPC Markup Language (EPML) [35]. The number of services in these processes is between 1 and 43.

The VM configurations are based on the current Amazon EC2[1] and are given in Table 2. In these experiments, the maximum number of VM instance is randomly generated in $[1..\frac{n}{2}]$ where n is the number of services. Another important parameter of the experiments is the maximum percentage of VM resource consumption. It is set to 90%. Table 3 shown the AM data input randomly generated.

Table 2: VM Configurations.

VM type	CPU (Cores)	CPU capacity (MIPS)	RAM (GB)	Bandwidth (MB/s)	Reliability (%)	Availability (%)	Compute price ($/hour)	Data transfer price ($/MB)
Micro	1	200	2	100	75	65	0.133	0.004
Small	2	400	4	1000	80	70	0.266	0.007
Medium	4	600	8	2000	85	75	0.532	0.015
Large	8	800	16	3000	90	80	1.064	0.031
xLarge	16	1000	32	4000	95	85	2.128	0.062
2xLarge	32	2000	64	8000	99	95	4.256	0.124

Table 3: AM informations.

Information	Range
AM number	[1..10]
Maximum instance number of AM	$[1..\frac{n}{2}]$
Length of AM	[1..100000]
AM' utilization price	[0.01..5]
Reliability requirement	[0..100]
Availability requirement	[0..100]
CPU requirement	[1..32]
RAM requirement	[0.1..64]
Bandwidth requirement	[0..8000]

4.2 Comparison with our former proposal

Our former approach presented in [17] consists of first determining the best mapping of AMs to services that minimizes the number of AMs while avoiding bottlenecks by assigning different AMs to services in parallel, and then, finding the best mapping of AMs to VMs that minimizes the overall communication cost.

[1] `http://aws.amazon.com/fr/ec2/`

As shown in Fig. 5, the proposed solution is cheaper than our former one. The average management cost decreases from 8.761$ to 4.160$. It reduces the cost by 52.517%. The reduction of the management cost can be explained by the fact that in our former approach, the deployment of a pre-determined number of AMs, where each AM can be used by several services that can be deployed on different VMs, leading to a high communication cost.

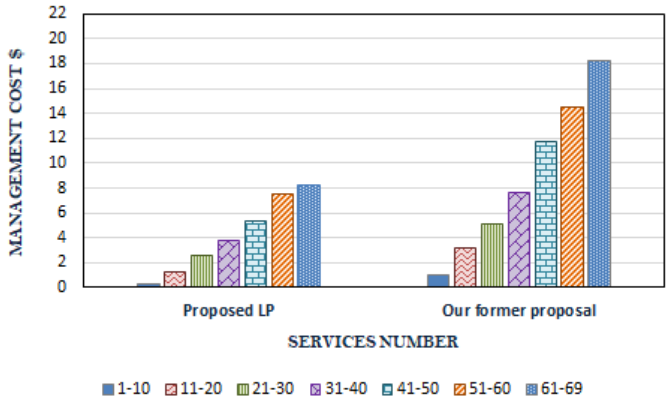

Fig. 5: The management cost using the proposed LP and our former proposal.

4.3 LP flexibility

4.3.1 Deadline constraint

The evaluation relies on adding a new constraint (24) which ensures that all services in the SBP must be executed before a deadline required by the customer.

$$makespan \leq deadline \tag{24}$$

The makespan of an SBP (*i.e.* the completion time of an SBP [36]) consists of two parts (see Equation 25), that is, the compute time ct of all services in the SBP (see Equation 26) and the transmission time tt among these services (see Equation 27).

$$makespan = ct + tt \tag{25}$$

$$ct = \sum_{i=1}^{|\mathbb{M}|} \sum_{j=1}^{|\mathbb{I}_{mi}|} \sum_{k=1}^{|\mathbb{V}|} \sum_{h=1}^{|\mathbb{I}_{vk}|} et_{ki} X_{ijkh} \tag{26}$$

$$tt = \sum_{p=1}^{|\mathbb{S}|} \sum_{i=1}^{|\mathbb{M}|} \sum_{j=1}^{|\mathbb{I}_{mi}|} \sum_{k=1}^{|\mathbb{V}|} \sum_{h=1}^{|\mathbb{I}_{vk}|} \frac{dt_{pi}}{bw_k X_{ijkh} (1 - Y_{ijp}) Z_{ijp}} \tag{27}$$

Table 4 shows that the proposed LP reaches the optimal solution in a few seconds (3.002s). However, solving the problem of minimizing the monetary cost of managing SBPs under a deadline constraint, the LP reaches the optimal solution but it requires higher computational time. The average computational time is increased from 3.002s to 8.356s and the average management cost increases from 4.162$ to 6.053$. To sum-up, we conclude that the proposed LP is flexible and tries to find solutions in a reasonable time, although by adding a new constraint, it is more difficult to obtain the optimal solutions.

4.3.2 Resource scaling

The evaluation is based on scaling up and down the maximum number of VM instances.

First, we scaled up the maximum number of VM instances from $\frac{n}{2}$ to n. As depicted in Table 5, our proposal reaches the optimal solution in a matter of seconds (3.484s). It reaches the optimal solution even better. Thus, the average management cost decreases from 4.162$ to 2.665$ due to the fact that, by doubling the number of VM instances, AMs are more likely to deploy in VMs with cheaper compute prices.

Second, we scaled down the number of VM instances to 1. As shown in Table 5, the LP finds the optimal solution in a reasonable time (2.122s) and the average management cost increases from 4.162$ to 7.617$.

Table 4: Experimental 1 results.

LP \ services#		1-10	11-20	21-30	31-40	41-50	51-60	61-69	Average
Proposed LP	Obj. fct ($)	0.356	1.247	2.612	3.803	5.397	7.503	8.216	4.162
	Time (s)	0.428	1.357	2.191	2.879	3.955	4.591	5.613	3.002
LP with deadline constraint	Obj. fct ($)	1.218	2.409	3.621	5.390	7.556	10.312	11.871	6.053
	Time (s)	0.313	3.145	5.602	8.269	10.778	12.539	17.847	8.356

Table 5: Experimental 2 results.

LP \ services#		1-10	11-20	21-30	31-40	41-50	51-60	61-69	Average
Proposed LP	Obj. fct ($)	0.356	1.247	2.612	3.803	5.397	7.503	8.216	4.162
	Time (s)	0.428	1.357	2.191	2.879	3.955	4.591	5.613	3.002
LP with resource scaling down	Obj. fct ($)	0.356	3.126	4.862	6.693	9.418	13.285	15.583	7.617
	Time (s)	0.183	0.826	1.569	2.061	2.562	3.412	4.247	2.122
LP with resource scaling up	Obj. fct ($)	0.356	0.838	1.583	2.327	3.232	4.912	5.409	2.665
	Time (s)	0.428	1.557	2.702	3.457	4.680	5.247	6.321	3.484

4.4 Conclusion

The experiment results demonstrate that the proposed LP is effective in terms of both quality and response time. Whenever the number of services is considerable, our proposal reaches the optimal solution in a reasonable time. We can conclude that it is by far much better than our former approach. In fact, a gain of 52.517% was measured on the management cost. Moreover, the LP can be easily adapted to cope with new constraints and different resource capacities.

5 Related work

In the literature, there have been several research works that aim to add autonomic management behaviors to cloud and distributed environments. In the following we give an overview of some of these works.

IBM is a pioneer in the field of autonomic computing that proposed an autonomic toolkit, which is a set of tools and technologies designed to permit users to add autonomic behavior to their systems. The authors in [21] presented all the needed steps to implement autonomic capabilities for resources. One of the main tools is the autonomic management engine that includes representations of the MAPE-K loop. Moreover, IBM suggested several tools to allow managed resources to create log messages using a standard format understandable by the MAPE-K loop. This is achieved using a touch-point that consists of a sensor and an effector. Moreover, an adapter rule builder is proposed to create specific rules in order to generate adaptation plans.

In [34], the authors proposed a framework for autonomic management of component-based applications. The different functionalities of a MAPE-K loop (*i.e.* Monitoring, Analysis, Planning, and Execution) are implemented as separate components, where each component is responsible for a single task. These components are attached to each component of an application for its self-management.

In [5], the authors presented an approach for improving the decision making process of a MAPE-K loop in order to self-adapt a component-based application. The authors equipped the analyzer component with sophisticated learning blocks, where the decision problem of the analyzer component is modeled as a Markov Decision Process with a finite set of states and actions. During each state transition, a reinforcement signal indicates to the proposed decision maker whether it choose the suitable action or not. In this work, each component of an application is self-managed by its own MAPE-K loop.

In [21], the authors proposed a framework for adding self-adaptation mechanisms to software systems. The proposed framework uses a model that represents an application as a graph. The graph consists of a set of nodes that represents components of an application and a set of arcs that represents interactions between components. The model is continuously adapted using a model manager. The latter model collects monitoring data from the model through probes. The collected data is analyzed using an evaluator that is able to detect violations and trigger adaptations. The appropriate adaptation plan is chosen using an adaptation engine and it is applied using an executor.

Authors in [4, 14, 18] adopted a decentralized approach to the autonomic management of SaaS applications. Each AM is dedicated to manage a part of an ap-

plication, and in most studies, they recommend the use of an AM per application service in order to improve management.

In their work [31], the authors focused on the coordination of multiple AMs in the cloud in order to efficiently manage the overall system. AMs are organized in a hierarchical structure, where the higher-level AMs have more authority over lower-levels AMs. These latter are responsible for allocating cloud resources, such as CPU and memory, to the web server in order to avoid SLA violations and improve its response time. The AMs communicate by exchanging predefined messages using a message broker.

De Oliveira et al. [33] proposed a framework for the coordination of AMs in the cloud. They presented two kinds of AMs known as "AAM" for Application AM and "IAM" for Infrastructure AM. Each application is managed by means of an AAM which is responsible for determining the best architectural configuration as well as the minimum number of VMs required to provide the best QoS under a certain workload. The IaaS cloud layer is managed by a single IAM which manages resource allocation in the infrastructure layer. The IAM holds a public and shared knowledge while each AAM maintains a private knowledge.

Several research works have been devoted to the issue of interaction and coordination of AMs. Broadly speaking, two methods are used for this. The first one splits the knowledge base of each AM into two parts: a public knowledge that is shared with the other AMs and a private knowledge for the AM [33]. The second method adds AMs that are in charge of coordination [16]. Our proposed approach is perfectly adapted to the first method where the coordination between AMs does not require additional AMs.

In the state of the art, there are other research works related to autonomic computing [11, 12, 26]. All of these approaches have been interested in modeling and implementing autonomic environments in a centralized or decentralized manner. In fact, some researchers have dedicated a centralized AM for the management of cloud and distributed systems without taking into account the large scale which may cause bottlenecks that could hinder the management efficiency. Other works tried to adopt the decentralization of AMs by (i) assigning an AM to each resource or (ii) random assigning of AMs to resources. These works do not take into account the management cost. However, the work presented in [29] addressed the optimal assignment of AMs to cloud resources. This approach does not take into account the diversity of QoS requirements and the heterogeneity of cloud resources. To the best of our knowledge, our proposed approach is the first that considers the problem of finding the optimal autonomic management of SBPs in the cloud while ensuring customers' QoS requirements and minimizes the management cost.

6 Conclusion

Managing service-based applications in the cloud involves using autonomic management capabilities in order to dynamically adapt services to changes. In this context, we proposed in this paper an approach for optimal autonomic management of SBPs in the cloud. The objective of our method was to minimize the management monetary cost while maintaining the QoS requirements. We solved the problem through a linear program-based optimizer. In addition, we extend the CloudSim simulator in order to validate our approach under realistic working

conditions. We evaluated our proposal using real datasets from IBM and the SAP reference model. The experiments results show the effectiveness, performance and flexibility of our approach.

As a future work, we intend to propose a near-optimal approach when dealing with large number of process services. In addition, we plan to test the proposed approach on a real cloud.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by the author.

Informed consent Informed consent was obtained from all individual participants included in the study.

Authorship contributions

Conception and design of study: L. Hadded, T. Hamrouni

Acquisition of data: L. Hadded

Analysis and/or interpretation of data: L. Hadded, T. Hamrouni

Drafting the manuscript: L. Hadded

Critical revision: T. Hamrouni

References

1. An architectural blueprint for autonomic computing. Tech. rep., IBM (2005)
2. Arunarani AR, Manjula D, Sugumaran V, Task scheduling techniques in cloud computing: A literature survey. Future Gener Comput Syst 91:407–415 (2019)
3. Bashar A, Modeling and simulation frameworks for cloud computing environment: A critical evaluation. In: International Conference on Cloud Computing and Services Science, pp 1–6 (2014)
4. Belhaj N, Lahmar IB, Mohamed M, Belaïd D, Collaborative autonomic management of distributed component-based applications. In: CoopIS (2015)
5. Belhaj N, Belaïd D, Mukhtar H, Self-adaptive decision making for the management of component-based applications. In: CoopIS (2017)
6. Booch G, Rumbaugh J, Jacobson I, Unified Modeling Language User Guide. Addison-Wesley Professional (2005)
7. Buyya R, Murshed M, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and Computation: Practice and Experience 14 (2002)

8. Buyya R, Ranjan R, Calheiros RN, Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In: International Conference on High Performance Computing Simulation, pp 1–11 (2009)
9. Casanova H, Legrand A, Quinson M, Simgrid: A generic framework for large-scale distributed experiments. In: International Conference on Computer Modeling and Simulation, pp 126–131 (2008)
10. Chen W, Xie G, Li R, Li K, Execution cost minimization scheduling algorithms for deadline-constrained parallel applications on heterogeneous clouds. Cluster Computing (2020)
11. Dehraj1 P, Sharma A, A review on architecture and models for autonomic software systems. The Journal of Supercomputing p 388–417 (2021)
12. Ebadifard F, Babamir SM, Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment. Cluster Computing pp 1573–7543 (2020)
13. Fahland D, Favre C, Koehler J, Lohmann N, Völzer H, Wolf K, Analysis on demand: Instantaneous soundness checking of industrial business process models. Data Knowl Eng p 448–466 (2011)
14. Florio L, Nitto ED, Gru: An approach to introduce decentralized autonomic behavior in microservices architectures. In: IEEE International Conference on Autonomic Computing (ICAC), pp 357–362 (2016)
15. Group OM, Business process model and notation (BPMN). Tech. rep. (2011)
16. Gueye SMK, De Palma N, Rutten É, Component-Based Autonomic Managers for Coordination Control. In: COORDINATION (2013)
17. Hadded L, Charrada FB, Tata S, Optimization and approximate placement of autonomic resources for the management of service-based applications in the cloud. In: CoopIS, pp 175–192 (2016)
18. Hadded L, Ben Charrada F, Tata S, Optimizing autonomic resources for the management of large service-based business processes. IEEE Transactions on Services Computing (2018)
19. Hadded L, Charrada FB, Tata S, Efficient resource allocation for autonomic service-based applications in the cloud. In: IEEE International Conference on Autonomic Computing (ICAC), pp 193–198 (2018)
20. IBM, IBM CPLEX Optimizer. `http://www.ilog.com/products/cplex/` (2021)
21. Jacob B, A Practical Guide to the IBM Autonomic Computing Toolkit. IBM Corporation, International Technical Support Organization (2004)
22. Jamshidi P, Ahmad A, Pahl C, Autonomic resource provisioning for cloud-based software. In: SEAMS (2014)
23. Juric MB, Business Process Execution Language for Web Services BPEL and BPEL4WS. Packt Publishing (2006)
24. Keller G, Teufel T, Sap R/3 Process Oriented Implementation. Addison-Wesley Longman (1998)
25. Kliazovich D, Bouvry P, Khan SU, Greencloud: a packet-level simulator of energy-aware cloud computing data centers. p 1263–1283 (2012)
26. Kosinska J, Zielinski K, Autonomic management framework for cloud-native applications. J Grid Comput 18(4):779–796 (2020)
27. Marino J, Rowley M, Understanding SCA (Service Component Architecture). Addison-Wesley Professional (2009)
28. Mell PM, Grance T, The NIST definition of cloud computing. Tech. rep. (2011)

29. Mohamed M, Megahed A, Optimal assignment of autonomic managers to cloud resources. In: SOLI (2015)
30. Mohamed M, Amziani M, Belaid D, Tata S, Mellit T, An autonomic approach to manage elasticity of business processes in the cloud. Future Generation Computer Systems pp 49 – 61 (2015)
31. Mola O, Bauer MA, Collaborative policy-based autonomic management: In a hierarchical model. In: International Conference on Network and Service Management (2011)
32. Núñez A, Vázquez-Poletti J, Caminero A, Castañé G, Carretero J, Llorente I, Icancloud: A flexible and scalable cloud infrastructure simulator. Journal of Grid Computing 10:185–209 (2012)
33. de Oliveira F, Ledoux T, Sharrock R, A framework for the coordination of multiple autonomic managers in cloud environments. In: SASO (2013)
34. Ruz C, Baude F, Sauvan B, Flexible adaptation loop for component-based soa applications. In: ICAS (2011)
35. Scheer AW, Thomas O, Adam O, Process Modeling Using Event-driven Process Chains. Wiley (2005)
36. Zhou J, Cao K, Cong P, Wei T, Chen M, Zhang G, Yan J, Ma Y, Reliability and temperature constrained task scheduling for makespan minimization on heterogeneous multi-core platforms. J Syst Softw 133:1–16 (2017)
37. Zhou J, Wang T, Cong P, Lu P, Wei T, Chen M, Cost and makespan-aware workflow scheduling in hybrid clouds. J Syst Archit 100 (2019)
38. Zhu Z, Tang X, Deadline-constrained workflow scheduling in iaas clouds with multi-resource packing. Future Gener Comput Syst 101:880–893 (2019)