

An Efficient Optimization Algorithm of Autonomic Managers in Service-Based Applications

Leila Hadded¹(✉), Faouzi Ben Charrada¹, and Samir Tata²

¹ Faculty of Science of Tunis, URAPAD,
University of Tunis El Manar, 2092 Tunis, Tunisia
hadded.leila@gmail.com, f.charrada@gnet.tn

² Institut Mines-Telecom, Telecom SudParis, UMR CNRS Samovar, Evry, France
Samir.Tata@mines-telecom.fr

Abstract. Cloud Computing is an emerging paradigm in Information Technologies that enables the delivery of infrastructure, software and platform resources as services. It is an environment with automatic service provisioning and management. In these last years autonomic management of Cloud services is receiving an increasing attention. Meanwhile, optimization of autonomic managers remains not well explored. In fact, almost all the existing solutions on autonomic computing have been interested in modeling and implementing of autonomic environments without paying attention on optimization. In this paper, we propose a new efficient algorithm to optimize autonomic managers for the management of service-based applications. Our algorithm allows to determine the minimum number of autonomic managers and to assign them to services that compose managed service-based applications. The realized experiments proves that our approach is efficient and adapted to service-based applications that can be not only described as architecture-based but also as behavior-based compositions of services.

Keywords: Cloud computing · Autonomic managers · Service-based applications · Optimization

1 Introduction

Cloud computing is a new computing paradigm that refers to a model for enabling convenient, on demand network access to a shared pool of configurable computing resources (e.g. servers, storage, applications and services). These resource can be rapidly provisioned and released with minimal management effort or service provider interaction [13]. In this paradigm, there are basically three levels for Cloud services' provision, which are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). At this later level, the effort is made to model, develop, deploy and manage applications and components/services that compose them. These applications

are also known as service-based applications (SBAs). Their service composition can be architecture-based (e.g. described in Service Component Architecture [12] or UML component diagram [4]) or behavior-based (e.g. described in Business Process Execution Language [11] (BPEL) or Business Process Model and Notation [16] (BPMN)).

Over the last years, autonomic computing got an increasing attention. It has been widely used in Cloud computing for dynamically adapting Cloud resources and service to changes in Cloud environments. Indeed, it aims at managing Cloud resources with minimal human intervention. Autonomic management usually relies on a MAPE-K (Monitor, Analyze, Plan, Execute and Knowledge) loop. This loop consists in collecting monitoring data from Cloud resources, analyzing them and producing series of planned changes to be executed on managed Cloud resources.

Managing SBAs according to principals of autonomic management, consists in determining and assigning MAPE-K loops to services that compose managed SBAs. To do that, two naive solutions can be considered. The first one consists in assigning one MAPE-K loop to a managed SBA. The second one consists in assigning one MAPE-K loop to each service of the managed SBA. It is obvious that the later solution is resource consuming while the former one may cause a bottleneck in managing SBAs. Consequently, it is of interest to optimize MAPE-K loopse consumption by minimizing the number of them while avoiding management bottlenecks.

When we visited the existing works on autonomic management and optimization of Cloud resources, we found out that they are not suitable to the considered problem of this paper. Indeed, on one hand, existing works on autonomic computing have been interested in modeling and implementing of autonomic environments without paying any attention to optimization. On the other hand optimization approaches are not adequate since they consider a number of resources known in advance.

In our previous works we have been interested in modeling, deployment and management of SBAs in Cloud environment [14, 23]. In this paper, we are interested in the optimization of number of autonomic managers (i.e. autonomic control loops) for SBAs. We propose a new algorithm consists of two steps. In the first step we determine all sets of services of a given SBA that can be run in parallel, which aims at determining the lower bound of the number of MAPE-K loops. In the second step, MAPE-K loops are determined, based on results from step one, and assigned to services of the managed SBA.

The rest of this paper is organized as follows. In Section 2, we present some preliminary notions on autonomic computing and we represent SBAs as graphs. Our proposed efficient algorithm for MAPE-K loops optimization is presented in Section 3. Experiments conducted on realistic data are detailed in Section 4. In Section 5, we present the state of art. Finally, we conclude the paper and we give directions for future works in Section 6.

2 Autonomic Management of Service-Based Applications

In this section, we present the background of our work, in which we aim at optimizing autonomic managers for the management of SBAs. We start with defining MAPE-K loop, then, we define service-based applications. After that, we show how these applications are represented as graphs. Finally, we present the problem of optimizing autonomic managers in SBAs.

2.1 The MAPE-K Control Loop

To achieve autonomic computing, IBM has suggested a reference model for autonomic control loops [1], which is called the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) loop as depicted in Fig. 1.

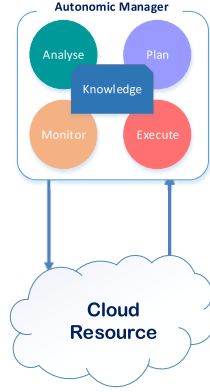


Fig. 1. Autonomic loop for a Cloud resource

This loop consists on harvesting monitoring data, analyzing them and generating reconfiguration actions to correct violations (self-healing and self-protecting) or to target a new state of the system (self-configuring and self-optimizing).

2.2 Service-Based Application

SBAs consists in composing a set of services using appropriate service composition specifications that can be architecture-based or behavior-based like. In the following we define these two types of compositions.

A SBA composed using an architecture-based composition can be described as a set of linked components. A component provides one or more services. It may consume one or several references, which are services provided by other components. As an example, we consider the online store example illustrated in Fig. 2 using a SCA assembly view. In the following sections, we use to this example in order to explain our concepts and motivate our work.

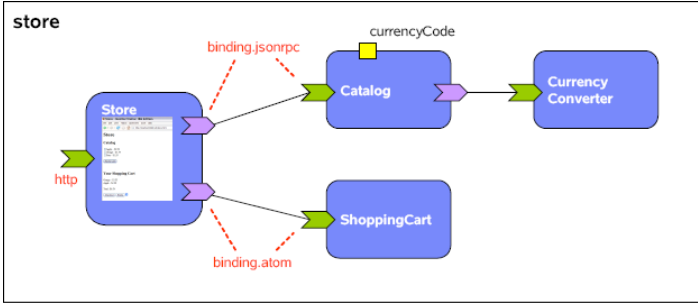


Fig. 2. Example of an on-line store (source [20])

The example is a composition of four services. The Store service provides the interface of the on-line store. The Catalog service which the Store service can ask for catalog items provides the item prices. The CurrencyConverter service does the currency conversion for the Catalog service. The ShoppingCart service is used to include items chosen from the Catalog service.

A SBA composed using a behavior-based specification can be described as a structured process which consists of a set of process nodes and transitions between them. A process node can be service, Or-Join, Or-Split, And-Split or And-Join. Fig. 3 depicts a BPMN business process of an online purchasing process of a clothing store.

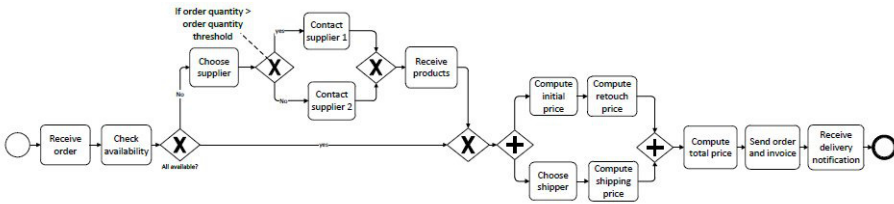


Fig. 3. Example of a SBA application modelled as a process

The customer sends a purchase order request with details about the required products and the needed quantity. Upon receipt of customer order, the seller checks product availability. If some of the products are not in stock, the alternative branch ordering from suppliers is executed. When all products are available, the choice of a shipper and the calculation of the initial price of the order are launched. Afterwards, the shipping price and the retouch price are computed simultaneously. The total price is then computed in order to send invoice and deliver the order. Finally, a notification is received from the shipper assuring that the order is already delivered.

The above-presented two types of compositions can be represented by graphs that we present in the following section.

2.3 SBA Graphs

The semantics of a graph that represents a SBA (called SBA graph) is described as follows. If s_1 and s_2 are nodes of graph and s_1 is connected to s_2 then the execution of s_2 follows the execution of s_1 or s_1 runs and references the services of s_2 during its execution. For instance, the later semantics reflects architecture-based compositions (e.g. SCA specification) while the former reflects behavior-based compositions (e.g. BPMN specification). Based on the above considerations, we can model a SBA like the one presented in Fig. 3 as a directed graph. Services, Or-Split, Or-Join, And-Split and And-Join nodes will be represented by graph nodes and connections/transitions between services will be represented by edges. Nodes are identified by an ID (a number).

Definition 1 (SBA graph). *A SBA graph is a 3 tuple $\langle S, E, v \rangle$ where:*

- *S is a set of services, Or-Split, Or-Join, And-Split and And-Join nodes composing the considered application (when the application is architecture-based S does not contain Or-Join, Or-Split, And-Split and And-Join nodes);*
- *$E \subseteq S \times S$ is the vertex connection set;*
- *v is the initial vertex of the graph.*

Fig. 4 represents the SBA graph of the SBA of Fig. 2 ($v=\text{Store}$).

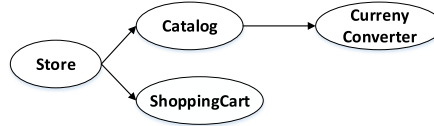


Fig. 4. The SBA graph of the on-line store

According to the semantics presented above, if the composition is behavior-based the execution of *Catalog* and *ShoppingCart* services are performed in parallel. Indeed, when the *Store* service is finished, *Catalog* and *ShoppingCart* services will be launched in parallel. Nevertheless, if the composition is architecture-based they may be run in sequence or in parallel. In fact, the *Store* service may reference both *Catalog* and *ShoppingCart* services in parallel or in sequence. We say in this later case that *Catalog* and *ShoppingCart* services may run in parallel.

2.4 Problem Statement

In this paper we present an approach for optimizing autonomic managers for the management of SBAs. Let's consider the example of Fig. 4 that presents a

SBA composed of four services. To determine the number of MAPE-K loops that can be assigned to the four services, two naive solutions can be considered. The first one, represented by Figure 5-(a), consists in considering one MAPE-K loop assigned to the four services that compose the managed SBA. The second solution, represented by Figure 5-(b), consists in considering four MAPE-K loops so that each one is assigned to one service. It is obvious that the later solution is resource consuming while the former one may cause a bottleneck in the management. It is, consequently, of interest to optimize MAPE-K loops consumption while avoiding management bottlenecks.

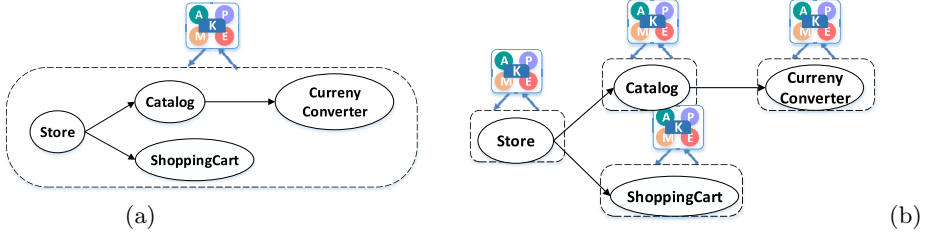


Fig. 5. Two naive solutions can be considered, (a) one MAPE-K loop assigned to the four services, (b) one MAPE-K loop assigned to each service

A solution, that can make a tradeoff between MAPE-K loops consumption, on one hand, and avoiding management bottleneck, on the other hand, would consist in considering two MAPE-K loops, one dedicated to *Store* and *ShoppingCart* services and one dedicated to *Catalog* and *Currency Converter* services. This later solutions minimize the number of used MAPE-K loops while not assigning one MAPE-K loop to more than one running service at a time.

Based on the above-mentioned illustrations, we can state the problem we tackle in this paper. Given a SBA graph, our objective is to determine the minimum number of MAPE-K loops needed to manage its services with the following requirement and assumption. Two services running in parallel should be provided with two different MAPE-K loops for their management to avoid the bottleneck problem. Two services that may run in parallel are considered running in parallel. This later assumption allows us to consider both types of composition semantics and cover different situations of service compositions whatever they are architecture-based or behavior-based.

3 Algorithm for an Efficient Optimization of MAPE-K Loops in SBAs

3.1 Approach Overview

In this section, we propose an algorithm for the optimization of autonomic managers (MAPE-K loops) in SBAs. This algorithm is based on four procedures called *Predecessor*, *LowerBound*, *ServiceRelatedParallelSets* and *AutonomicLoopsAssignment*. The *Predecessor* procedure consists in determining

for each service the number of its predecessors. *This procedure is used when the application is behavior-based* where a service begins execution only when all its predecessors have finished execution. The *LowerBound* procedure consists in determining a set of sets of services that satisfy the following property. Services that belong to one set can be run in parallel. The *ServiceRelatedParallelSets* procedure consists in determining for each service the set of services that can be run in parallel with it. *AutonomicLoopsAssignment* consists in assigning to each service a MAPE-K loop which is different from loops that are already assigned to services to be run in parallel with it. In the following we present these four above-mentioned procedures.

3.2 Predecessor Procedure

The *Predecessor* procedure, presented in Algorithm 1, takes as input a SBA graph. It returns an array containing for each service the number of its predecessors. Initially, the number of predecessors of each service is equal to zero (see Algorithm 1, lines 1-3). For each service s , the number of its predecessors is incremented by 1 if there is a service s_i successor of s_i (see lines 4-6).

Algorithm 1. Predecessor procedure

Require: $\langle S, E, v \rangle$: SBA graph

Ensure: *Predecessors*: array containing for each service the number of its predecessors

```

1: for all  $s \in S$  do
2:   Predecessors[ID of  $s$ ]  $\leftarrow 0$ ;
3: end for
4: for all  $(s_i, s) \in E$  do
5:   Predecessors[ID of  $s$ ]  $\leftarrow$  Predecessors[ID of  $s$ ] + 1;
6: end for
```

3.3 LowerBound Procedure

The *LowerBound* procedure, presented in Algorithm 2, takes as input a SBA graph and an array containing for each service the number of its predecessors. It returns a set of sets of services that can be run in parallel and the maximum cardinality of its elements which constitutes a lower bound number of MAPE-K loops. The initial vertex of the SBA graph v is assigned to an initial set (see Algorithm 2, line 1). The *LowerBound* procedure is to be executed while the current set of services that can be run in parallel is not empty and is not a subset of a set that is already made in a previous iteration (see line 24). The current set of services composed of services, which are successors of services of the previous set where all its predecessor services are already treated in the previous sets (see lines 10-11). Otherwise, the number of predecessors is decremented by 1 (see line 13). If the current set is not already made in a previous iteration, then it is added to the set of sets (see lines 18-19). The lower bound number is possibly updated (see lines 20-22). This number is the maximum number of services that can be

run in parallel. When the application is architecture-based the current set of services is composed of services, which are successors of services of the previous set (doesn't exist a test to check for each service s_j that all its predecessors are treated (see line 10)).

Algorithm 2. LowerBound procedure

Require: $\langle S, E, v \rangle$: SBA graph
Require: *Predecessors*: array containing for each service the number of its predecessors
Ensure: *ParallelSets*: set of sets of services that can be run in parallel
Ensure: *lbn*: the lower bound number

```

1: CurrentParallelSet  $\leftarrow \{v\}$ ;
2: ParallelSets  $\leftarrow \{\textit{CurrentParallelSet}\}$ ;
3: lbn  $\leftarrow 1$ ;
4: repeat
5:   PreviousParallelSet  $\leftarrow \textit{CurrentParallelSet}$ ;
6:   CurrentParallelSet  $\leftarrow \emptyset$ ;
7:   for all  $s_i \in \textit{PreviousParallelSet}$  do
8:     for all  $s_j \in S$  do
9:       if  $(s_i, s_j) \in E$  then
10:        if Predecessors[ID of  $s_j$ ] = 1 then
11:          CurrentParallelSet  $\leftarrow \textit{CurrentParallelSet} \cup \{s_j\}$ ;
12:        else
13:          Predecessors[ID of  $s_j$ ]  $\leftarrow \textit{Predecessors}$ [ID of  $s_j$ ] - 1;
14:        end if
15:      end if
16:    end for
17:  end for
18:  if  $\nexists \text{ set } s.t. (\text{set} \in \textit{ParallelSets} \text{ and } \textit{CurrentParallelSet} \subseteq \text{set})$  then
19:    ParallelSets  $\leftarrow \textit{ParallelSets} \cup \textit{CurrentParallelSet}$ ;
20:    if  $|\textit{CurrentParallelSet}| > \textit{lbn}$  then
21:      lbn  $\leftarrow |\textit{CurrentParallelSet}|$ ;
22:    end if
23:  end if
24: until (CurrentParallelSet =
     $\emptyset$  or  $\exists \text{ set } s.t. (\text{set} \in \textit{ParallelSets} \text{ and } \textit{CurrentParallelSet} \subseteq \text{set})$ )
  
```

The needed MAPE-K loops for a given SBA graph may be greater than the lower bound number. To give an example of such situation let's consider the example of Fig. 6 when the application is architecture-based as depicted in Section 2.3 if s_1 and s_2 are nodes of graph and s_1 is connected to s_2 then s_1 runs and references the services of s_2 during its execution.

Applied to this later example, the *LowerBound* procedure produces the following results:

- *ParallelSets*: $\{\{s_1\}, \{s_2, s_3\}, \{s_4, s_5\}, \{s_5, s_6\}, \{s_6, s_3\}, \{s_3, s_5\}\}$
- *lbn* = 2

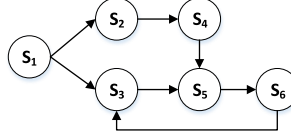


Fig. 6. Example of a SBA graph of a SCA-based Application

With respect to the semantics of applications specified in SCA, First, s_5 and s_6 may run in parallel. Second, s_6 and s_3 may run in parallel. Third, s_3 and s_5 may run in parallel. To satisfy these requirements, on one hand, and to assign different MAPE-K loops for services running in parallel, on the other hand, it is obvious that the number of needed MAPE-K loops for this example is equal to 3, while the lower bound number is equal to 2. Therefore, we need additional computing based the result of the *LowerBound* procedure to determine the number of needed MAPE-K loops for a given SBA and their assignment. This is the objective of the following algorithms.

3.4 *ServiceRelatedParallelSets* Procedure

The *ServiceRelatedParallelSets* procedure, presented in Algorithm 3, takes as input a SBA graph and a set of sets of services that can be run in parallel. It returns for each services s the set of services which can be run in parallel with it. This set is the union of all sets that belong to *ParallelSets* and that contain s (see Algorithm 3, lines 5-9).

Algorithm 3. *ServiceRelatedParallelSets* procedure

Require: $\langle S, E, v \rangle$: SBA graph
Require: *ParallelSets*: set of sets of services that can be run in parallel
Ensure: *ServiceRelatedParallelSets* : array of $\langle serviceIndex, serviceSet \rangle$

- 1: $i \leftarrow 1$;
- 2: **for all** $s \in S$ **do**
- 3: *ServiceRelatedParallelSets*[i].*serviceIndex* $\leftarrow ID$ of s ;
- 4: *ServiceRelatedParallelSets*[i].*serviceSet* $\leftarrow \emptyset$;
- 5: **for all** $set \in ParallelSets$ **do**
- 6: **if** $s \in set$ **then**
- 7: *ServiceRelatedParallelSets*[i].*serviceSet* \leftarrow
 ServiceRelatedParallelSets[i] $\cup (set - \{s\})$;
- 8: **end if**
- 9: **end for**
- 10: $i \leftarrow i + 1$;
- 11: **end for**

3.5 *AutonomicLoopsAssignment* Procedure

The *AutonomicLoopsAssignment* procedure, presented in Algorithm 4, takes as input a SBA graph and an array containing for each service the set of services

that can be run in parallel with it. It returns an array containing for each service the MAPE-K loop assigned to it and the number of MAPE-K loops used for the managed SBA represented by the input SBA graph. *AutonomicLoops* is an array that will contain for each service s , the number of the MAPE-K loop assigned to it. Initially, the elements of this array are equal to zero, which means that loops are not yet assigned to services (see Algorithm 4, lines 1-3). MAPE-K loops assignment begins with the service whose related parallel set has the biggest cardinality. Consequently, the array *ServiceRelatedParallelSets* is sorted in decreasing order according to the cardinality of sets of its elements (see line 4). For each service s (see lines 6-14), the variable *currentLoop* is initialized with the value 1. This is a tentative to assign the loop number 1 to the s (see line 7). If this *currentLoop* isn't already assigned to a service that belongs to the set of services that can be run in parallel with s , then *currentLoop* is assigned to s (see line 10). Otherwise, *currentLoop* is incremented by 1. This is done to try assigning the next loop to s (see line 12). This computing is repeated until assigning a loop to s (see line 14). The number of autonomic loops needed to manage the given SBA is then computed (see lines 15-17).

Algorithm 4. AutonomicLoopsAssignement procedure

Require: $\langle S, E, v \rangle$: SBA graph
Require: *ServiceRelatedParallelSets* : array of $\langle serviceIndex, serviceSet \rangle$
Ensure: *AutonomicLoops*: array containing for each service the MAPE-K loop assigned to it
Ensure: *numberAutonomicLoops*: number of MAPE-K loops

```

1: for all  $i \in \{1, 2, \dots |S|\}$  do
2:   AutonomicLoops[ $i$ ]  $\leftarrow 0$ ;
3: end for
4: sortDecreasingOrderOfCardinalityOfSets(ServiceRelatedParallelSets);
5: numberAutonomicLoops  $\leftarrow 0$ ;
6: for all  $i \in \{1, 2, \dots |S|\}$  do
7:   currentLoop  $\leftarrow 1$ ;
8:   repeat
9:     if  $\nexists s$  s.t. ( $s \in ServiceRelatedParallelSets[i].serviceSet$ 
       and AutonomicLoops[ID of  $s$ ] = currentAutonomicLoop) then
10:      AutonomicLoops[ServiceRelatedParallelSets[ $i$ ].serviceIndex]  $\leftarrow$ 
        currentLoop;
11:    else
12:      currentLoop  $\leftarrow$  currentLoop + 1;
13:    end if
14:  until  $\nexists s$  s.t. ( $s \in ServiceRelatedParallelSets[i].serviceSet$ 
    and AutonomicLoops[ID of  $s$ ] = currentAutonomicLoop)
15:  if currentLoop > numberAutonomicLoops then
16:    numberAutonomicLoops  $\leftarrow$  currentLoop;
17:  end if
18: end for

```

Applied to our running example presented in Fig. 4:

- The *LowerBound* procedure gives the following results:

- *ParallelSets*: $\{\{Store\}, \{Catalog, ShoppingCart\}, \{Currency Converter\}\}$
- $lbn = 2$ which is the cardinality of the second element in the *ParallelSets* $\{Catalog, ShoppingCart\}$
- The *ServiceRelatedParallelSets* procedure gives the following results:

Services	Store	Catalog	ShoppingCart	Currency Converter
ServiceRelatedParallelSets	$\langle 1, \emptyset \rangle$	$\langle 2, \{ShoppingCart\} \rangle$	$\langle 3, \{Catalog\} \rangle$	$\langle 4, \emptyset \rangle$

- The *AutonomicLoopsAssignment* procedure gives the following results:
 $numberAutonomicLoops = 2$

Services	Catalog	ShoppingCart	Store	Currency Converter
AutonomicLoops	1	2	1	1

Applied to our running example presented in Fig. 6:

- The *ServiceRelatedParallelSets* procedure gives the following results:

Services	S_1	S_2	S_3	S_4	S_5	S_6
ParallelSets	$\langle 1, \emptyset \rangle$	$\langle 2, \{S_3\} \rangle$	$\langle 3, \{S_2, S_5, S_6\} \rangle$	$\langle 4, \{S_5\} \rangle$	$\langle 5, \{S_3, S_4, S_6\} \rangle$	$\langle 6, \{S_3, S_5\} \rangle$

- The *AutonomicLoopsAssignment* procedure gives the following results:
 $numberAutonomicLoops = 3$

Services	S_3	S_5	S_6	S_2	S_4	S_1
AutonomicLoops	1	2	3	2	1	1

4 Experiments

In service research field, there are two types of compositions of services: behavior-based and architecture-based compositions. Behavior-based compositions of services are generally sparse graphs where nodes represent services and operators and links represent dependencies between services and operators. Architecture-based compositions of services can be sparse or dense graphs where nodes represent services and links represent dependencies between services.

To evaluate our algorithm for the optimization of MAPE-K loops for SBAs in the cloud, we have considered two datasets, one for architecture-based compositions and one for behavior-based compositions. At the best of our knowledge, there is no public and open source dataset for architecture-based compositions of services. Therefore, in Section 4.1, we give the results of experiments performed on a realistic dataset based on randomly generated graphs, which represented architecture-based compositions of services. But in Section 4.2 we give results related to a real dataset from IBM that contains 560 BPMN business process. All the computation times are achieved on intel® Core™ i5 CPU a 2.53 GHz 2.53GHz, RAM 4Go.

As we will explain in Section 5, at the best of our knowledge none of the existing approaches tackles the problem of optimizing the number of autonomic managers while avoiding the bottleneck problem. Consequently, we are not able to cover any comparison with an existing approach. Therefore, for both experiments, we studied the time complexity of our algorithms and the quality of their results in the sense of the closeness of results to the lower bound numbers results of the *LowerBound* procedure.

4.1 Experiments on Architecture-Based Compositions

To consider a realistic dataset, we have covered different graphs to represent different types of compositions of services. In fact, our generated graphs are constructed as follows. For each graph of order n we consider to represent a SBA that should be connected. Consequently, this later should contain at least $n - 1$ edges (when it is a tree). To cover different types of SBAs with the same order n , we have considered different graphs with different number of edges starting from $n - 1$ edges until $3.2 * (n - 1)$ (i. e. 320% of $(n - 1)$). In fact, we did not consider additional graphs with more edges, since we found out that beyond $2.2 * (n - 1)$, the lower bound number is n . Then the number of needed loops for a graph of order n , in this case, is n .

For our experimentation, we have varied graphs' order 10 times from 10 to 100. Real datasets, such as the IBM DataSet [7], show that 99% of service-based applications are within this order range (less than 100). In addition for each order n , we considered 12 densities (from 100% of $(n - 1)$ to 320% of $(n - 1)$) and for each density, we considered 10 randomly generated graphs. In total, we have considered 1200 generated graphs. Table 1 summarize the characteristics of our dataset.

Table 1. Characteristics of generated graphs (the values presented in this table are the number of edges of the graph and n is the number of nodes)

$\begin{matrix} \% \text{ of } n-1 \\ n \end{matrix}$	100%	120%	140%	160%	180%	200%	220%	240%	260%	280%	300%	320%
10	9	11	13	14	16	18	20	22	23	25	27	29
20	19	23	27	30	34	38	42	46	49	53	57	61
30	29	35	41	46	52	58	64	70	75	81	87	93
40	39	47	55	62	70	78	86	94	101	109	117	125
50	49	59	69	78	88	98	108	118	127	137	147	157
60	59	71	83	94	106	118	130	142	153	165	177	189
70	69	83	97	110	124	138	152	166	179	193	207	221
80	79	95	111	126	142	158	174	190	205	221	237	253
90	89	107	125	142	160	178	196	214	231	249	267	285
100	99	119	139	158	178	198	218	238	257	277	297	317

As depicted in Fig. 7, that presents the evolution of percentage of the lower bound number with respect to the service number using different densities, when the number of edges for a graph of order n is beyond $2.2 * (n - 1)$, the lower bound number is n . Then the number of needed loops is n . Therefore, we limited

our experimentations' analysis for quality of results, of the *AutonomicLoopsAssignment* procedure, to graphs with number of edges are starting from $n - 1$ up to $2.2 * (n - 1)$, for graphs of order n .

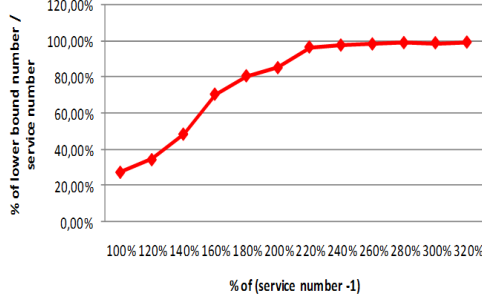


Fig. 7. Evolution of % of lower bound number / service number using different densities

Complexity. It is obvious that the time complexity of Algorithms 1, 3 and 4 is polynomial whereas the theoretical time complexity of Algorithm 2 is exponential (i.e. $o(2^n)$ where n is the order of the considered graph). In fact, the time complexity is equal to $o(|ParallelSets|)$ which is bounded by $o(2^n)$. Nevertheless, from a practical point of view, the execution time of our algorithm is reasonable. For the 1200 considered graphs the execution time does not exceed 0.24 seconds.

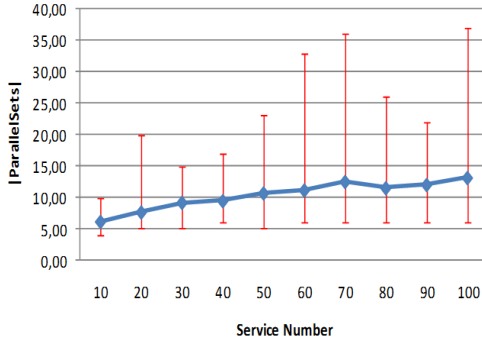


Fig. 8. Evolution of ParallelSets'cardinality using different type of graphs

As it is shown in Fig. 8, the number of sets of services that can be run in parallel ($|ParallelSets|$) is small with respect to graphs' order. For instance, the number of parallel sets for graphs of order 100 does not exceed 37. In addition, the curve of $|ParallelSets|$ is linear with a very low slope.

Quality. According to our assumptions depicted in Section 2.4, when the number of MAPE-K loops result of the *AutonomicLoopsAssignment* procedure is equal to the lower bound number result of the *LowerBound* procedure, then this former number is optimal. Applied to our dataset, our algorithm gives excellent results since it obtained optimal results, in the above sense, for 94% of the considered graphs. Beyond optimal results, let's analyze the quality of non-optimal ones (6% of the considered graphs). The quality of a given result is measured by the difference, in terms of number of assigned loops, between the lower bound obtained by the *LowerBound* procedure, on one hand, and the number of loops obtained by the *AutonomicLoopsAssignment* procedure on the other hand. The lower this difference is the better it is.

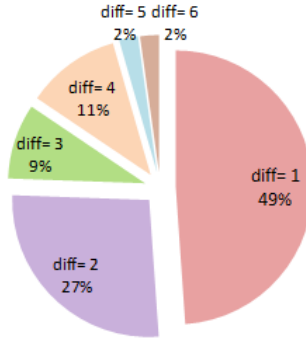


Fig. 9. Average of difference percentages

As it is shown in Fig. 9, the difference for non-optimal results (which constitute themselves 6% of the whole results) does not exceed 6 loops, where for 49% among them, the difference is equal to one. Note as it is shown in Fig. 6, the needed assigned loops for a given SBA can be greater than the lower bound obtained by the *LowerBound* procedure.

4.2 Experiments on Behavior-Based Compositions

To evaluate our Algorithms, we also used an IBM DataSet that contains 560 BPMN business processes, which represent a real dataset of behavior-based compositions of services available in the IBM WebSphere Business Modeler tool. A process node can be startEvent, endEvent, task, exclusiveGateway, parallelGateway, inclusiveGateway or subprocess. The number of process nodes varies between 7 and 533 while the number of task varies between 2 and 106.

Complexity. From a practical point of view, the execution time of our algorithm on the IBMs dataset is reasonable. In fact, for the 560 considered business processes, the execution time does not exceed 0.1 second.

Quality. According to our assumptions depicted in Section 2.4, when the number of MAPE-K loops result of the *AutonomicLoopsAssignment* procedure is equal to the lower bound number result of the *LowerBound* procedure, then this former number is optimal. Applied to the IBMs dataset, our algorithm gives excellent results since it obtained optimal results, with respect to the above sense, for 100% of the considered business processes.

5 Related Work

In Cloud and distributed environments, there are several research works related to autonomic computing as well as optimization of Cloud resource consumption. At the best of our knowledge, these proposals treat the two areas separately. In the following, we give an overview of some of these works.

5.1 Autonomic Computing

One of the pioneers in the Autonomic Computing field is IBM that proposed a dedicated toolkit [18]. In this work, authors gave the IBMs definition of Autonomic Computing as well as the needed steps to define autonomic resources for the management of components. The proposed toolkit is a collection of technologies and tools that allows a user to develop autonomic behavior for his/her systems. One of the basic tools is the Autonomic Management Engine that includes representations of the MAPE-K loop that provides self-management properties to managed resources.

Beside the IBMs work, Buyya et al. proposed a conceptual architecture to enhance autonomic computing for Cloud environments [5]. The proposed architecture is basically composed of a SaaS web application used to negotiate the SLA between the provider and its customers, an Autonomic Management System (AMS) located in the PaaS layer. The AMS incorporates an Application scheduler responsible of assigning Cloud resources to applications. It also incorporates an Energy efficient scheduler that aims to minimize the energy consumption of all the system. The AMS implements the logic for provisioning and managing virtual resources.

In [21], authors proposed an Autonomic Network-aware Meta-scheduling (ANM) architecture capable of adapting its behavior to the current status of the environment. This work is based on a Grid Network Broker (GNB) that represents the autonomic network-aware meta-scheduler. GNB chooses the most appropriate resource to run jobs. An autonomic loop is implemented to adjust the scheduling task to improve job completion times and resources utilization. Whenever the selected resource did not respond to the required QoS, other resources are checked until a suitable resource is found.

In [19], authors introduced a framework that tackle management and adaptation strategies for component-based applications. In their approach, the authors separate Monitoring, Analysis, Planning and Execution concerns by implementing each one of them as separate components that could be attached to a managed component.

de Oliveira et al [15] proposed a framework for self-management of systems which focuses on the coordination of autonomic managers in the Cloud. The authors proposed an architectural model for autonomic managers coordination that meets the Cloud architectural constraints from the perspective of loose coupling and information hiding. Two kinds of autonomic managers are presented in this paper. The first kind consists in managing the applications at the SaaS layer, which is called Application Autonomic Manager (AAM). The second kind consists in managing the IaaS layer, which is called Infrastructure Autonomic Manager (IAM). In this paper, authors proposed to assign one AM to each managed system that can be one application or the whole infrastructure.

All these autonomic computing approaches have been interested in modeling and implementing of autonomic environments without making any effort for optimizing autonomic managers used for the management of applications. In contrast, in our work, we propose a novel approach to optimize autonomic resources used for the management of service-based applications.

5.2 Optimization of Cloud Resources

In their work [6], Chaisiri et al. proposed an optimal Cloud resource provisioning (OCRP) algorithm for the management of virtual machines. This work can help the consumer to decide whether to purchase reserved or on-demand instances of Cloud computing resources in each time slot with the objective of reducing the total provisioning cost.

Babu et al. [3] introduced a generic algorithm to allocate virtual machines optimally in Cloud environments. Initially they proposed to assign each application to a virtual machine and compute the remaining capacity. Therefore they apply a genetic algorithm in order to have an optimal allocation to the virtual machines, with the maximum remaining capacity.

Yusoh et al. [9], presented a service deployment strategies for efficient execution of Composite SaaS applications in the Cloud. The objective was to determine which services should be assigned to which virtual machines. To achieve this objective, authors took inter-service communication and parallelism among services into consideration. They proposed an approach to minimize communication costs by assigning interrelated services in the same virtual machine and increasing the potential execution parallelism by assigning two independent services in different virtual machines when application is modeled as a Directed Acyclic Graph (DAG).

In [8], authors presented a novel approach to schedule elastic processes in the cloud. They define a system model and an optimization model which is aiming at minimizing the total leasing cost for Cloud-based computational resources. The problem addressed is to determine which services should be assigned to which virtual machines.

In [10], authors proposed an algorithm for scheduling of workflow applications in geographically distributed Clouds taking into account interdependence between workflow steps and permits to assign each tasks to Cloud resources in

order to minimizing cost and execution time according to the preferences of the user.

In [17,22], authors presented a Particle Swarm Optimization (PSO) based algorithm to optimize the schedules tasks in workflow applications among Cloud services that takes computation cost and data transmission cost into account in order to minimize the execution cost when application is modeled as a DAG.

Arya et al. [2], reviewed the basis workflow scheduling algorithms that are important for cloud environments. Different methods are used in these algorithm (i.e., Particle Swarm Optimization, Heuristic based Genetic Algorithms, etc.) and several factors are considered such as the execution time, resource utilization, cost optimization, etc.

At the best of our knowledge, in the works related to optimization of Cloud resource mainly those we cite above the number of Cloud resources is assumed to be known in advance. While one can imagine adapting these proposed algorithms to optimize autonomic managers' consumption, by considering an autonomic manager as a cloud resource, these works can not address our objective. In fact, in these work, the number of Cloud resources is assumed to be known in advance, while in our work the number of autonomic managers is not known in advance. In addition, some of these works don't address applications with dependency relationships. The work we present in this paper is novel in the sense that (1) it tackles the problem of optimization at the SaaS level (particularly for SBPs) while considering applications with dependency relationship and (2) it tackles the problem of autonomic computing when the number of autonomic managers isn't known in advance.

6 Conclusion and Future Work

In this paper, we present a novel approach to optimize autonomic managers used for the management of service-based applications. Our approach consists in (1) determining all sets of services for a given SBA that can be run in parallel, which aims to determine the lower bound number of MAPE-K loops, and (2) assigning MAPE-K loops to services. The proposed algorithms are of acceptable time complexity from a practical point of view. The execution time on graphs of different types and orders does not exceed 0.24 seconds. To evaluate the quality of results, we have conducted more than 1200 of experiments on graphs of a realistic dataset. Experiments results show that our algorithm has an excellent behavior for architecture-based compositions and for those based on behavior.

The work we achieved is very promising and several perspectives are under study. Among others, we aim, in the short term, at considering estimations on execution time of service when determining and assigning MAPE-K loops to services. Another possible extension, if such information is not available, is an online approach that consists in determining and assigning, within a time window, MAPE-K loops to services during their execution. In a longer term, we aim at considering an approach to determine and assign MAPE-K components, rather than loops, to services. Since monitors, analyzers, planners, executors, and

knowledge bases component may not be used in the same way in the management of services, we can envisage to assign some MAPE-K components to several services running in parallel and dedicate some others to one service at a time depending on their usage.

References

1. An architectural blueprint for autonomic computing. Tech. rep., IBM (2005)
2. Arya, L., Verma, A.: Workflow scheduling algorithms in cloud environment - a survey. In: Engineering and Computational Sciences (RAECS) (2014)
3. Babu, K.D., Kumar, D.G., Veluru, S.: Optimal allocation of virtual resources using genetic algorithm in cloud environments. In: Proceedings of the 12th ACM International Conference on Computing Frontiers (2015)
4. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language User Guide. Addison-Wesley Professional (2005)
5. Buyya, R., Calheiros, R.N., Li, X.: Autonomic cloud computing: open challenges and architectural elements. CoRR (2012)
6. Chaisiri, S., Lee, B.S., Niyato, D.: Optimization of resource provisioning cost in cloud computing. IEEE Transactions on Services Computing (2012)
7. 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. (2011)
8. Hoenisch, P., Schuller, D., Schulte, S., Hochreiner, C., Dustdar, S.: Optimization of complex elastic processes. IEEE Transactions on Services Computing (2015)
9. Huang, K.C., Shen, B.J.: Service deployment strategies for efficient execution of composite saas applications on cloud platform. Journal of Systems and Software (2015)
10. Juhnke, E., Dornemann, T., Bock, D., Freisleben, B.: Multi-objective scheduling of BPEL workflows in geographically distributed clouds. In: IEEE International Conference on Cloud Computing (CLOUD) (2011)
11. Juric, M.B.: Business Process Execution Language for Web Services BPEL and BPEL4WS, 2nd edn. Packt Publishing (2006)
12. Marino, J., Rowley, M.: Understanding SCA (Service Component Architecture). Addison-Wesley Professional (2009)
13. Mell, P.M., Grance, T.: The NIST definition of cloud computing. Tech. rep. (2011)
14. Mohamed, M., Amziani, M., Belaïd, D., Tata, S., Melliti, T.: An autonomic approach to manage elasticity of business processes in the Cloud. Future Generation Computer Systems (2014)
15. de Oliveira, F., Ledoux, T., Sharrock, R.: A framework for the coordination of multiple autonomic managers in cloud environments. In: IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO) (2013)
16. (OMG), O.M.G.: Business process model and notation (BPMN). Tech. rep. (2011)
17. Pandey, S., Wu, L., Guru, S., Buyya, R.: A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: 24th IEEE International Conference on Advanced Information Networking and Applications (2010)
18. Redbooks, I., Organization, I.B.M.C.I.T.S.: A Practical Guide to the IBM Autonomic Computing Toolkit. IBM Corporation, International Technical Support Organization (2004)

19. Ruz, C., Baude, F., Sauvan, B.: Flexible adaptation loop for component-based SOA applications. In: 7th International Conference on Autonomic and Autonomous Systems ICAS (2011)
20. The Apache Software Foundation: Getting started with Tuscany. <http://tuscany.apache.org/getting-started-with-tuscany.html>
21. Tomás, L., Caminero, A.C., Rana, O., Carrión, C., Caminero, B.: A gridway-based autonomic network-aware metascheduler. *Future Gener. Comput. Syst.* (2012)
22. Wu, Z., Ni, Z., Gu, L., Liu, X.: A revised discrete particle swarm optimization for cloud workflow scheduling. In: International Conference on Computational Intelligence and Security (CIS) (2010)
23. Yangui, S., Tata, S.: The spd approach to deploy service-based applications in the cloud. *Concurrency and Computation: Practice and Experience* (2014)

On the Move to Meaningful Internet Systems: OTM 2015
Conferences

Confederated International Conferences: CoopIS,
ODBASE, and C&TC 2015, Rhodes, Greece, October
26-30, 2015. Proceedings

Debruyne, C.; Panetto, H.; Meersman, R.; Dillon, T.;
Weichhart, G.; An, Y.; Ardagna, C.A. (Eds.)

2015, XXV, 678 p. 236 illus. in color., Softcover

ISBN: 978-3-319-26147-8