Inteligencia Artificial 25(69) (2022), 183-200
doi: 10.4114/intartif.vol25iss69pp183-200

INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

# Fault Tolerance for Composite Cloud Services: A MAS-based Novel Approach

Ouassila Hioual[1,2,A], Sofiane Mounine Hemam[1,B], Ouided Hioual[1,C], Rania Mordjane[1], Nessrine Bouhlala[1]
[1] Abbes Laghrour University of Khenchela, Algeria
[2] LIRE Laboratory, Abdelhamid Mehri University of Constantine 2, Algeria
[A] ouassila.hioual@gmail.com, [B] sofiane.hemam@gmail.com, [C] ouided.hioual@gmail.com

**Abstract** Several Cloud services may be composed in order to respond quickly to the users' needs. Unfortunately, when running such a service some faults may occur. The outcome of fault control is a big challenge. In this paper, the authors propose a new approach based back recovery and multi-agent planning methods. The proposed architecture based MAS (Multi-Agent System) is composed of two main types of Agents: a Composition Manager Agent (CMA) and a Supervisor Agent (SA). The role of the CMA is to create a set of plans as an oriented graph where the nodes are the Cloud services and the valued arcs represent the composition order of these services. This agent saves checkpoints (nodes) in a stable memory so that there are at least one possible path. However, the SA ensures that the running plan is working properly; otherwise, it informs the CMA to select another sub-plan. Experimental results show the performance of the proposed approach in term of different metrics such as response time, cost and total cost of different alternatives of cloud services.

**Keywords**: Fault tolerance, Back recovery, Check point, Composition plans, Cloud service, Multi Agents System, Minimal path, Composite service.

## 1 Introduction

The Cloud computing is an emerging computing technology which is, nowadays, used by many of the other technologies such as big data, block-chain technology, deep learning [1]. It is a distributed computing model that uses resources on the Internet. It is designed to run large-scale applications. This is a cost-effective way to achieve the Service Level Agreement (SLA) and Quality of Service (QoS) according to application demand [2].

Cloud Computing offers advanced services that we can use at any time. Thereby, it represents a trend change, because we can use the computing and storage power, offered by a provider, in a distributed system, instead of acquiring new hardware and software for obtaining functionalities in an isolated system. Thus, Cloud computing can be categorized as a computing solution in which required technology or services allow users to access computing resources on demand [3].

Cloud services are considered as partial solutions that must be composed in order to provide to users a virtual service. This composite service should be automated and must be dynamic in order to respond quickly to the needs of users [4]; [5]. Unfortunately, in some cases, and during the execution of such a cloud service, it is expected that many failures will occur. This situation will extend the expected time to respond to customer requests and deplete cloud resources, on the one hand. On the other hand, customers will not get their services on time. Thus, for the Cloud, failures will lead to the loss of resources and money [6]. Therefore, fault tolerance measures are vital for cloud computing due to its high dynamicity, huge number of the resources and possibilities of failures in cloud resources [7]. Thus, the outcome of fault control is a big challenge because of its direct influence on the reliability, safety, productivity and operational efficiency of a composite Cloud service. Our work context is part of this challenge. To ensure the availability and reliability of services, fault tolerance has become one of the major concerns of any system. In order to minimize the impact of failures on the system and ensure correct task execution, they must be anticipated and managed [3]. There are a few number of newly developed methods for fault tolerance, which have focused on fault detection dimension.

Various fault tolerance techniques can be used at either task level or workflow level to resolve failures [8]. Fault tolerance (FT) techniques can, typically, be categorized into two classes: reactive fault tolerance techniques and proactive fault tolerance techniques. Reactive fault tolerance approaches focus on reducing the effect of failures on task execution if there is any resource failure. It is implemented using check-points and restart mechanisms [9]. Reactive fault tolerance techniques have the advantage of reducing the impact of failures on a system when failures have effectively occurred. However, it is also possible to anticipate failures and proactively take action before failures occur in order to minimize failure impact on the system and application execution [10].

The check pointing technique is used when doing task scheduling, the check-points are inserted to identify fault incidence [4]. The different checked points are selected through an oriented graph, which represent different plans. These plans can represent the different alternatives of a composition solution. In the literature, several approaches have been proposed to solve the fault tolerance problem in Cloud Computing, including multi-agent planning approach. An agent is a computer system that is capable of making decisions independently and interacting with other agents through cooperation, coordination, and negotiation [11]. Using an agent-based approach, characteristics associated with intelligent behaviors of agents such as interacting socially through cooperation, coordination, and negotiation can be built into clouds [11], and it has proved to be effective for distributed applications.

Our research problem is a part of the fault tolerance in a cloud environment while the execution of a composite cloud service. Our goal is to propose an approach based back recovery and minimal paths selection from a graph. The latter represents the different possible execution plans for such a composite service. In our context, back recovery based check pointing consists in capturing enough data during fault-free distributed execution of tasks using a Multi-Agent System (MAS). Furthermore, when a fault appears we must choose another sub-plan to complete the execution of the composed service.

The rest of the paper is organized as follows: Section 2 presents a brief literature review. Section 3 introduces the proposed approach and its functioning over an illustrative example. Section 4 gives the experimental results and their analysis. Finally, section 5 makes some conclusions and indicates future works.

## 2 Related works

Many researchers have been devoted to the field of fault tolerance in cloud environments. Indeed, several works have, recently, been proposed on autonomic fault tolerance which use different techniques such as back recovery based check pointing. There is also a wide set of works in intelligent cloud computing that tries to make clouds more intelligent by adopting intelligent agents to automate the interactions among clouds and between consumers and cloud [11]. The present section is divided into two subsections. The first one introduces the recently proposed fault-tolerant strategies, in a Cloud Computing environment, using different techniques. On the other hand, the second subsection will make more highlights on some works that used the MAS for Cloud Computing purposes.

### 2.1 Fault tolerance approaches in the cloud computing

In a Cloud Computing environment, fault tolerance is a critical requirement because it allows the system to provide good performance in the presence of one or more system component failures. In this sub section, we enumerate some promising techniques and approaches that tackle with this open issue.

In [12], a reactive fault tolerance technique, called Virtual Machine- μ (VM- μ) Checkpoint framework, was proposed. Copy on Write- Pre-save algorithm and check-pointing was used to protect VMs against transient errors. In addition, in order to save, in advance, the checkpoints of the tasks running in the VMs, a cache memory was used. In the proposed algorithm in-memory incremental checkpoints was taken so that restoration can be done in-place.

The authors in [13] presented an overview of workflow temporal checkpoint selection. They proposed a temporal checkpoint selection strategy to deal with business workflows. Zhang et al. [14] introduced a novel system which analyses automatically logs files to predict any failure warning signals. The authors of [14] analyzed the proposed solution by comparing them with machine learning approaches based on logical traces. The obtained results demonstrated the high capability of the proposed system to predict failures within complex systems. In [15], the authors presented an earlier strategy that included a non-preemptive scheduling with task migration algorithm. The proposed solution included an algorithm that migrates the aborted task and starts the execution at the point where the latest checkpoint was saved. This leads to better performance and achieves Quality of Services (QoS), according to authors of [15]. However, the proposed method had a major drawback of starting again the task in another virtual machine that greatly increases the execution time of the migrated task.

In [16], the authors proposed a new online fault detection model based on Support Vector Machnie-Grid (SVM-Grid). The model was constructed based on SVM and its key parameters were optimized by the grid method. The authors proposed methods for fine-tuned prediction and updating fault sample databases. This made it possible to improve prediction performance and decrease time cost. In addition, they have compared the proposed model with Back Propagation (BP) and Learning Vector Quantization (LVQ) methods. The experimental results have clearly shown that the proposed model has better accuracy and lower time cost than the two other methods.

In [17], the authors proposed a strategy that implicated how to calculate the optimal number of checkpoints based on failure event distribution. From the fact that various parameters like check-pointing overload, time delay have an impact on the cloud system, an adaptive algorithm was designed. In order to implement the model, the parameters that have been considered were user request of multiple task, number of jobs, checkpointing cost, checkpointing position, probability of failure event, execution time and wall clock time. Experimental results have shown a better adequacy of the system for large scale applications.

The authors in [18] have introduced a predictive fault module based on the analysis of the available data in the Cloud infrastructure. This module works in correlation with the other modules dedicated to load balancing and scheduling which have been introduced in a previous work of the same authors [19]. In addition, they have studied the correlations between the values of smart attributes and the probability of failure of a hard disk relative to a given server. According to the authors, the fault tolerance' management module can prevent the main controller from assigning a virtual machine to a server that presented a risk of failure [18].

In [20], the authors proposed a self-adaptive monitoring approach for cloud computing systems. This latter is composed of three steps. The first one conducted correlation analysis between different metrics, and monitors selected important ones, which represented the others and reflected the running status of a system. The second step characterized the running status with Principal Component Analysis (PCA). Finally, in the third step, the authors have dynamically adjusted the monitoring period based on the estimated anomaly degree and a reliability model. In addition, to evaluate their proposal, they have applied the approach in their open-source TPC-W (Transaction Processing Performance Council- transactional Web) benchmark Bench4Q (a QoS-oriented e-commerce benchmark) deployed in their real cloud computing platform OnceCloud. The experimental results demonstrated that the proposed approach could adapt to dynamic workloads, accurately estimate the anomaly degree, and automatically adjust monitoring periods. Thereby, the proposed approach in [20] have effectively improved the accuracy and timeliness of anomaly detection in an abnormal status, and efficiently lowered the monitoring overhead in a normal status.

In [21], a fault detection and diagnosis approach was proposed for multi-tier web application in infrastructure cloud computing. The approach is composed of three steps. The authors, first, proposed the Fuzzy One-Class Support Vector Machine (FOCSVM) model to detect abnormal data based on the decisive boundary. Then, the exponentially weighted moving average chart, which monitors the decisive boundary value of FOCSVM model, was applied to detect faults for multi-tier web application. Finally, the authors have applied Random Forest ranking feature algorithm in order to determine if suspicious metrics were the root of faults.

Authors of [7] proposed an Adaptive Fault Tolerant Resource Allocation (AFTRA) approach, as a proactive fault tolerance measure. The proposed approach was implemented based on an application that was self-adaptive with respect to its current position or state in its state space and the applications or tasks that were submitted. According to the authors, their approach effectively ensured procedures that were done automatically according to the situation and thus providing better reliability to critical tasks under temporal and resources constraints. In [3], the authors have provided a fuzzy-based method to detect and prepare an appropriate response to the error tolerance, after having given a detailed analysis of the nature of the error. In addition, in order to increase the error tolerance and load balancing when the error occurs, the authors have used the requesting a task re-execution and migration techniques through the checkpoint. According to authors in [3], the migration technique overlaps with time, so check-point use can avoid re-execution as well as tasks as much as possible.

## 2.2 SMA-uses for Cloud Computing purposes

Multi-agent systems are a set of autonomous agents interacting in a common environment in which, they cooperate to accomplish a given task. By the joint use of cloud computing and multi-agent systems, many common problems can be distinguished and lot of qualities can be determined [22]. According to [23], the number of studies that can be found on the state of the art relating Cloud Computing with agent technology is actually quite low. However, everything is changing, but even more slowly. Moreover, in recent years, we often find studies and applications focused on this field. Indeed, we can find, in the literature, a considerable number of proposals that use agent technology to solve some issues that exist in Cloud Computing. In [24], the authors proposed an agent-based cloud system for cloud service discovery and reservation. It consists of a set of agents that consult an ontology to select relevant information about cloud services. The aim was the selection of the best cloud service that has the best Price and Time slot Negotiation (PTN).

The work presented in [25] proposes a new mechanism that deploys various intelligent agents to reduce the cost of virtual machines and resource allocation complexity in cloud computing environment. This research proposed a new Agent based Automated Service Composition (A2SC) algorithm. The latter is a comprising of request processing and automated service composition phases. According to the authors, it is not only responsible for searching comprehensive services but also considers reducing the cost of virtual machines which are consumed by on-demand services only.

In [26], the authors presented an Intelligent Multi-Agent Reinforcement Model (IMARM) for optimizing cloud resource allocation. The proposed model uses check pointing and VM migration mechanisms to ensure both fault tolerance and load balancing. To determine the system reaction, sensor agents report, periodically, system failure status, energy consumption, and workload [26]. On another side, the authors used Q-Learning in order to indicate which action will be performed in each system state. The main parameters used by IMARM are the weight of the virtual machines, the total energy consumption, and the quantum time.

The proposed model in [23] is based on agent technology and is able to distribute computational resources throughout the entire Cloud Computing environment. It allows the distribution of its complexity, and the improvement of its associated computational costs. In addition, the authors claim that the proposed model allows the decision- making process to be carried out right where the information is gathered, on the base that provides local knowledge, which has made it possible to design agile control processes based on uncertain information, prior knowledge, and the interaction among similar agents [23].

Based on the literatures reviewed, attempting to use MAS in Cloud Computing issues has been ongoing but adequate attention was not given to the issue of fault tolerance of composite cloud services. In the present work, we deal with the fault tolerance problem in a cloud environment by taking into consideration the situation when some faults may occur while running a composite cloud service. For that, we try to take advantages from the above-cited works such as check pointing techniques and MASs from the fact that they provide appropriate services to the system. Indeed, our architecture is an agent-based one because agents are capable of solving problems independently and may collaborate with one another to achieve objectives. In our architecture, we assume that a Composition Manager Agent (CMA) has a complete knowledge of all services deployed in the Cloud. The Supervisor Agent (SA) ensures that the running plan is working properly and in the case of a failure, it informs the CMA to apply the back recovery technique and to select another minimal sub-plan using Dijkstra algorithm [27]. In the following, we present and analyze the proposed approach.

## 3 Proposed approach and architecture

In this section, we present the proposed approach and the system architecture on which this latter can function.

### 3.1 Proposed Architecture

Dependability is an important aspect in any distributed system. This is why a Fault Tolerance (FT) mechanism becomes necessary to ensure certain aspects. The primary categories of faults in Cloud include network fault, physical fault, process fault and service execution fault [28]. Our contribution consists in proposing an approach allowing masking the failure of a component when running a composite Cloud service. Therefore, we deal with faults belonging to the last of the categories mentioned above.

The duality between agent, as an autonomous and adaptive entity, and MAS, as a decentralized and cooperative organization, offers a very privileged framework for solving a given problem such as FT in Cloud Computing. It is for these reasons we have chosen to use MASs in our work.

The architecture of our system (see. Figure 1) is an agent-based one which is inspired from a previous work [4] with some improvements. The two main agents of the architecture are as follows:

- The Composition Manager Agent (CMA) that conceives and creates the different plans that represent the alternative plans necessary to have the appropriate composition of services responding to the user request. These plans constitute an oriented graph enabling it to define the ideal path (a sub-plan). It looks for the minimal path (the plan having a cheapest cost), it transfers this last to the Supervisor Agent (SA) that is responsible for the proper functioning of the chosen minimal path. During the execution of the chosen minimal path, the CMA will save, each time, a service as a checkpoint provided that from this service there is at least one other minimal path. In addition, the saved service must be available at the time of the fault occurs.

- In addition, the Supervisor Agent (SA) whose role is to control and ensure the proper execution of the chosen plan (path). In the case of a fault appearance, it informs the CMA in order to take the necessary measures.

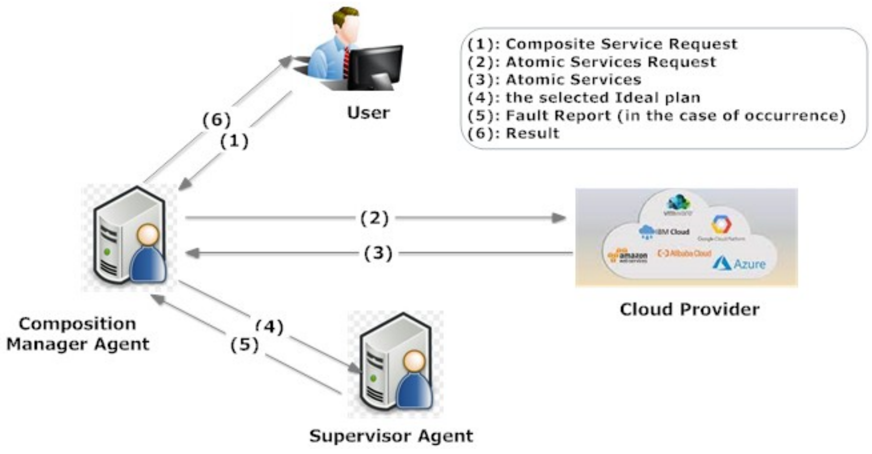

(1): Composite Service Request
(2): Atomic Services Request
(3): Atomic Services
(4): the selected Ideal plan
(5): Fault Report (in the case of occurrence)
(6): Result
User
(6)
(1)
(2)
(3)
Cloud Provider
Composition Manager Agent
(4)
(5)
Supervisor Agent
IBM Cloud
amazon
Alibaba Cloud
Azure

Figure 1. The proposed architecture of our system.

The Composition Manager Agent and the Supervisor Agent are cognitive agents because they can anticipate, predict the future and memorize information. These two agents are intelligent by them selves, that is to say that they can carry out a certain reasoning to choose their actions. In addition, the CMA and SA have a knowledge base comprising various information related to their areas of expertise and to the management of interactions between them and their environment. This type of agents (i.e. cognitive agents) are generally intentional, that is, they have explicit goals and plans to accomplish their goals.

## 3.2 Functioning of the Proposed Approach

In this subsection, we will illustrate how our approach works through a sequence diagram illustrated in Figure 2. We use the check pointing technique based on the back recovery, and the multi-agent planning by choosing the minimum paths.

According to the sequence diagram (Figure 2), the CMA, after receiving a request for a composite service, transmits it to the Cloud provider. Thus, it creates the different service composition plans of which each plan represents a possible solution to meet customer needs. Then, it chooses the minimal path, among them, and sends it to the Supervisor Agent (SA). The latter ensures the proper functioning of the selected plan (path). During this time, the CMA saves, periodically, a checkpoint. When a fault will be detected by the SA, it informs the CMA in order to choose the adequate sub-plan (minimal sub-path) from the most recent checkpoint, and so on until the composite service will be successfully executed.

We assume that the different plans of a composite service constitute an oriented graph (see Figure 3). The root of this later is the initial cloud service that represents the start of component services for all alternative plans of a given composite service. The nodes of this graph represent the component cloud services. An arc between two nodes is an oriented arc and represents the succession link, that is to say that an arc connecting the node *a* to the node *b* means that: the cloud service *b* will be executed after the good execution of the cloud service *a*. Therefore, according to our approach, a checkpoint is a node from which we can find a minimal sub-plan (cf. Figure 3). The bows have values that represent weighted values between the Cloud service cost, estimated time of its execution and its reliability. These values are used by the CMA in order to select the ideal path (initial plan or a sub-plan after a fault detection). Moreover, these three values are not of the same type, so they need to be normalized. To explain this point, we refer to the following three tables: Table 1, Table 2 and Table 3.

To estimate the weight of the arc between the two services S1 and S2 for example (cf. Figure 3), we use these values that are illustrated in Tables 1, 2 and 3 and apply formula (1):

$$weight(S1 - S2) = \frac{\frac{C(S1)+R(S1)+T(S1)}{3} + \frac{C(S2)+R(S2)+T(S2)}{3}}{2} \qquad (1)$$

Such as:

C (Si): the estimated value of the Cost of a service.
R (Si): the estimated value of the Reliability.
T (Si): the estimated value of the execution Time.

Table 1: Normalization of a service cost

| **Cost of a service ($)** | [50-100[ | [100-150[ | [150-200[ | P>=200 |
|---|---|---|---|---|
| **Normalized value** | 0.1 | 0.2 | 0.3 | 0.4 |

Table 2: Normalization of a service reliability

| **Number of users** | [1k-2k[ | [2k-3k[ | [3k-4k[ | Nb>=4k |
|---|---|---|---|---|
| **Reliability** | 0.4 | 0.3 | 0.2 | 0.1 |

Table 3: Normalization of the estimated execution time of a service

| **Execution time** | [0.1-.03[ | [0.3-0.5[ | [0.5-0.7[ | T>=0.7 |
|---|---|---|---|---|
| **Estimated value** | 0.1 | 0.2 | 0.3 | 0.4 |

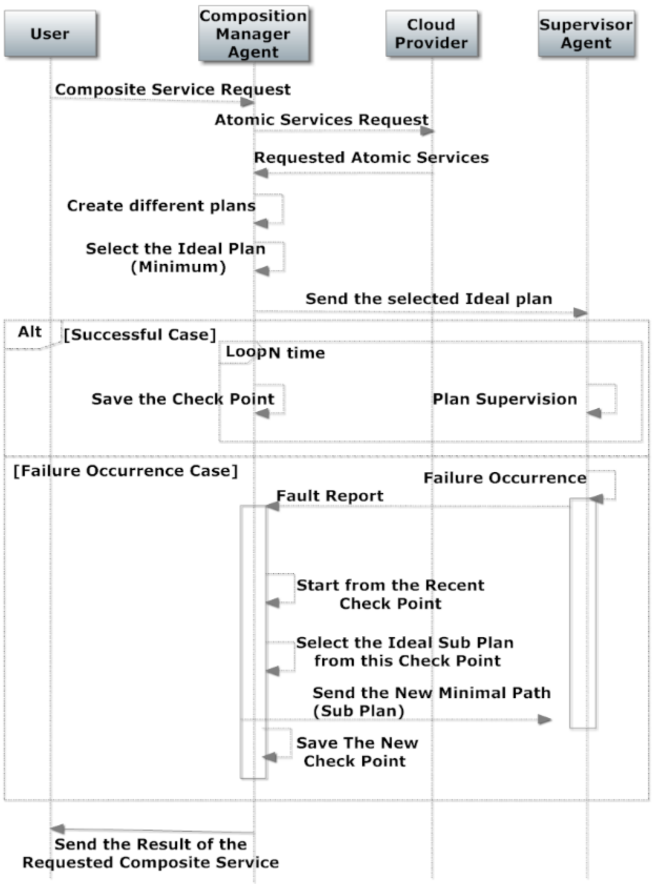
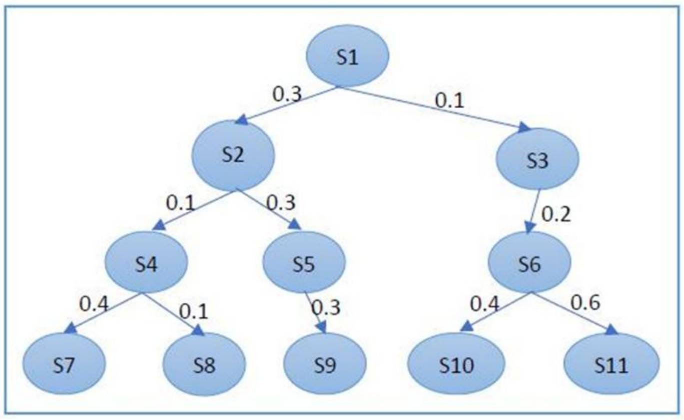

User
Composition Manager Agent
Cloud Provider
Supervisor Agent
Composite Service Request
Atomic Services Request
Requested Atomic Services
Create different plans
Select the Ideal Plan (Minimum)
Send the selected Ideal plan
Alt
[Successful Case]
LoopN time
Save the Check Point
Plan Supervision
[Failure Occurrence Case]
Failure Occurrence
Fault Report
Start from the Recent Check Point
Select the Ideal Sub Plan from this Check Point
Send the New Minimal Path (Sub Plan)
Save The New Check Point
Send the Result of the Requested Composite Service

Figure 2. Sequence diagram which represents the functioning of the proposed approach.



.

Figure 3. Plan of a composite service.

## 3.3 Composition Manager Agent Behaviour

We assume that the Composition Manager Agent (CMA) has full knowledge of all services deployed in the Cloud. It is responsible for conceiving and creating the various plans for a composite service in the form of a directed graph to meet a user demand. In this paper, we assume that the CMA has a set of plans (at this stage, it is considered that the list of plans is *predefined*. However, in a future work this list will be *conceived* and *created* dynamically according to the user's request (i.e. the composite cloud service to be executed)). In addition, the CMA saves, periodically, a checkpoint by using the stable memory mechanism [29] [30]. This is done provided that: (i) since this point there is at least a minimal sub-plan and, (ii) the service is available at the time of the failure production. The data to be saved for each checkpoint are:

- The service ID ;
- The service size (the size of a service to be run in a cloud resource);
- The service FileSize (the size of the file service before execution);
- The service OutputSize (the size of the service execution result);
- And, the service state.

In the case of a failure, it applies the back recovery technique and selects another minimal sub-path, which has minimal cost. The failed task is restarted from the most recent checkpoint rather than from the beginning. We illustrate the behaviour of this agent by the activity diagram of Figure 4.

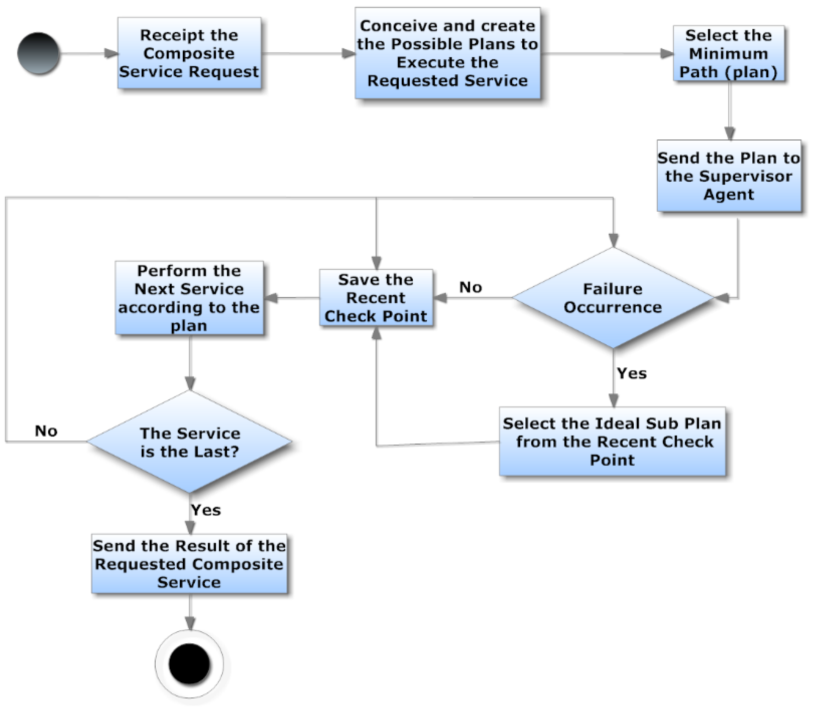

Receipt the Composite Service Request
Conceive and create the Possible Plans to Execute the Requested Service
Select the Minimum Path (plan)
Send the Plan to the Supervisor Agent
Failure Occurrence
No
Save the Recent Check Point
Perform the Next Service according to the plan
Yes
Select the Ideal Sub Plan from the Recent Check Point
No
The Service is the Last?
Yes
Send the Result of the Requested Composite Service

Figure 4. Activity diagram of the Composition Manager Agent.

### 3.3.1 Selection of the Minimal Plan (path):

To select the minimal plan (the minimal path), the CMA uses Dijkstra's algorithm, and the latter makes it possible to find the shortest path from a node initial.

Table 4: Progress of Dijkstra Algorithm on the previous plan

| S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 | S11 | S12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| **0** | 0.5 | 0.6 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| **0** | 0.5 | 0.6 | 0.8 | 1.2 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| **0** | 0.5 | 0.6 | 0.8 | 1.2 | 2.1 | 1.8 | ∞ | ∞ | ∞ | ∞ | ∞ |
| **0** | 0.5 | 0.6 | 0.8 | 1.2 | 2.1 | 1.8 | 0.9 | ∞ | ∞ | ∞ | ∞ |
| **0** | 0.5 | 0.6 | 0.8 | 1.2 | 2.1 | 1.8 | 0.9 | 3.2 | 4.2 | ∞ | ∞ |
| **0** | 0.5 | 0.6 | 0.8 | 1.2 | 2.1 | 1.8 | 0.9 | 3.2 | 4.2 | ∞ | 1.83 |
| **0** | 0.5 | 0.6 | 0.8 | 1.2 | 2.1 | 1.8 | 0.9 | 3.2 | 4.2 | 2.9 | 1.83 |

To perform this composite service (cf. Figure 5), we take the path A (S1, S2, S4, S8) because it has the minimum value (0.9). Therefore, the path A (plan A) represents the best plan for this composite service because it has the minimum of Total Cost (TC). If a failure occurs at service level S2, we return to the most recent recovery point (S1), and we choose another minimal plan: we take plane B (S1, S3, S7, S12).

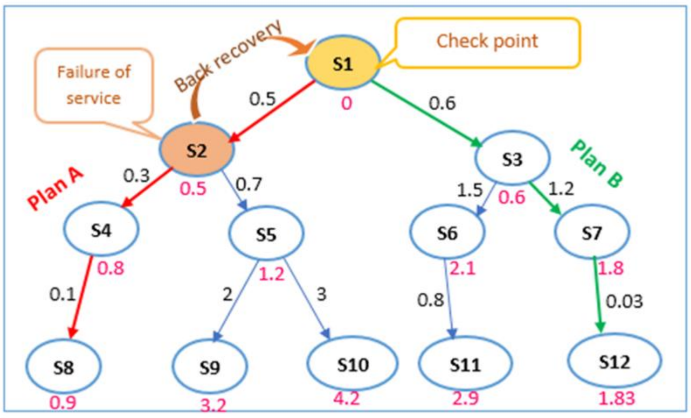


Figure 5. Different plans of a composite service in a failure case

### 3.3.2 Registration of Checkpoints:

The pseudo code below (cf. Algorithm 1) is used to show how the CMA proceeds to save checkpoints and use them in the event of a failure. This code is executed repeatedly to return a reliable composite service as result.

**Algorithm 1 : Registration_Check_point**

```
1 Input : Chosen_path, Tab1, Failure_report : Dynamic vector ;
2 Output :  Check_point : service ; Chosen_path : Dynamic vector ;
3  Begin
4   Check_point=Chosen_path [service1] ;
5   Tab1=Check_point;
6   While (The execution of the composed service is not complete) Do
7       If (Chosen_path [service i]≠Failure_report [serviceN]) Then
8           Check_point=Chosen_path [service i] ;
9           Tab1=Tab1+Check_point ;
10      Else
11          If (Chosen_path [service i]== Failure_report [serviceN]) Then
12              serviceJ=Check_point ;
13              Funct_shortest_path(liste_path(serviceComp),serviceJ) ;
14          End If ;
15      End If ;
16  End While ;
17  Display Tab1 content  ;
18  Display the new Chosen_path;
19 End.
```

## 3.4 Supervisor Agent Behaviour

The SA role is to control and ensure the proper functioning of the chosen minimal plan. If a fault occurs and it is detected, it must react immediately and informs the CMA by a fault report, which contains useful information such as at what level the fault occurred. The diagram below is a better illustration of the behavior of the SA (Cf. Figure 6).

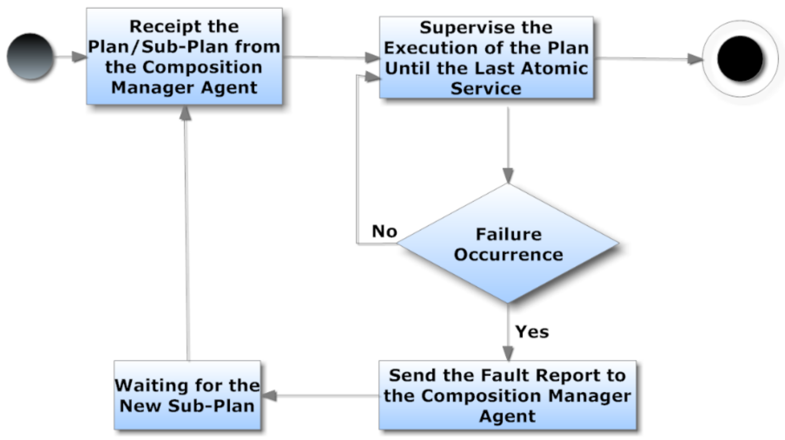

Receipt the Plan/Sub-Plan from the Composition Manager Agent
Supervise the Execution of the Plan Until the Last Atomic Service
Failure Occurrence
No
Yes
Send the Fault Report to the Composition Manager Agent
Waiting for the New Sub-Plan

Figure 6. Activity diagram of the Supervisor Agent

Concerning how the supervision of the plan/sub-plan is carried out: as already mentioned in section 3.2, each atomic service, component of a chosen plan/sub-plan, is characterized by an estimated execution time. When the SA receives the plan/subplan from the CMA, for each atomic service it initializes its clock with the estimated execution time of the latter. Once the time is up, it checks if the atomic service has completed successfully, in which case it resets its clock with the execution time of the next service. Otherwise, it informs the CMA and waits for the new sub-plan.

## 3.5 Communication inter agents

Each agent has a communication module that manages the reception and interpretation of messages from the other agent. This management concerns the linguistic medium of message transmission, i.e. the expression and interpretation of a message. We have chosen, as the message expression language, the FIPA ACL (Foundation for Intelligent Agent Foundation Agent Communication Language) communication language. Therefore, according to FIPA ACL, an agent sends a message in two scenarios:

1. If it has received a message that needs to be processed, it then builds a message to respond to the one it has received and sends it to the sender of the latter;
2. If it performs an action whose effects consist of sending one or more messages, then it constructs these messages and sends them to the specified recipient(s).

In our context, the content of messages, sent by the CMA to the SA, is the description of the plan to be executed in terms of a suite of cloud services that make up the requested service. Consequently, the messages sent by the SA to the CMA contain the list of erroneous services

## 4 Simulation and results

In this section, we first introduce an illustrative scenario to explain the proposed approach through a real case. Then, in Sub-section 4.2, we present the results obtained using CloudSim simulator and discuss the advantages of the proposed approach.

## 4.1 An illustrative scenario

The purpose of this scenario is to illustrate two points: (i) The progress of our approach on a real case, and (ii) the interaction between the different agents of the proposed architecture.

For this, we have chosen the example of a service consisting of an organized trip. The CMA proceeds to the representation of atomic (component) services in the form of a directed graph (cf. Figure 7). First, the CMA requests the set of services, which are the components of the requested composite service (organized trip) from a cloud provider. These services are organized according to their chronological order.

The CMA applies Dijkstra's algorithm and obtains the values shown in the graph of the figure 7, in order to choose the least expensive path. We obtain the following minimal path:

**Organized trip -> Destination 1 -> Airplane ticket -> Hotel 5***.

Each time, the CMA keeps a service as a checkpoint provided that since this service there is at least one other path. At the same time, the SA watches over the correct operation of the chosen plan. We suppose that a failure has occurred at level 3 in the airplane ticket reservation service: (cf. Figure 7).

**Organized trip -> Destination 1 -> Airplane ticket -> Hotel 5***.

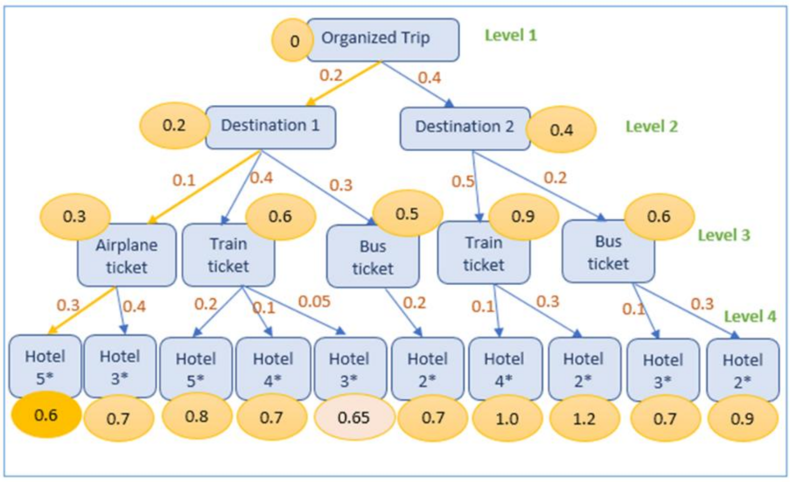


Figure 7. The directed graph of the different plans of the requested service.

In this case, the SA sends a fault report to the CMA as follows:

(**Level 3: Service: Airline ticket reservation**)

So, the CMA after having received an error message, it applies the Back Recovery mechanism to the most recent checkpoint. According to the example, it is: (**Level 2: Service: Destination 1**)

The CMA reapply Dijkstra's algorithm to find a new sub-plane that is different from where the failure occurred. (cf. Figure 8). Therefore, the new sub-plan is:

***Destination1 -> Train ticket -> Hotel***

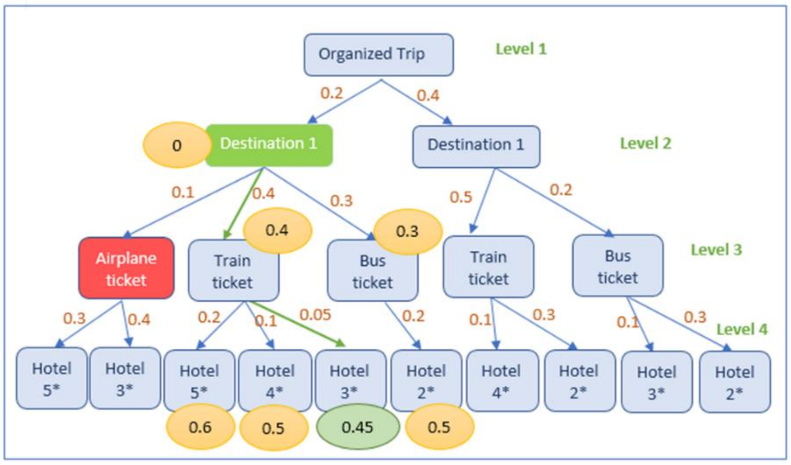


Figure 8. The graph after the re-application of Dijkstra's algorithm

## 4.2 Experimental results and discussion

In this sub-section, our approach is validated through simulation by using Cloudsim [31]. CloudSim allows simulation of Cloud computing systems. In addition, it is an extensible simulation toolkit enabling the modelling application provisioning environments. In order to implement our experiments, we had extended several classes of Cloudsim. The concerned classes are Cloudlet, VMs and DataCenter. Indeed, we have extended:

(1) The cloudlet class in order to support the characteristics of our services, which are the execution time (T), the cost (C), the reliability (R) and the Total Cost (T.C);

(2) The VMs class to specify the memory, storage, and bandwidth of virtual machines, and;

(3) The DataCenter class to model the service core infrastructure (hardware and software) offered by resource provider. At this level, for the execution of the services, we link each service (cloudlet) to a virtual machine.

To implement our two agents classes (CMA and SA), we have used JADE [32] environment. As these classes inherit from the Agent class of the library (jade.jar), we must implement and configure each agent separately. After including the necessary libraries and compiling, these two agents' classes must be included in the Cloudsim library. To do this, in the Arguments tab of CMA and SA agents, respectively, we write the following codes:

```
-gui jade.Boot CMA.org:cloudbus.cloudsim.cloud.ControllerManagerAgent
-gui jade.Boot SA.org:cloudbus.cloudsim.cloud.SuppervisorAgent
```

The experiments are performed on a machine with an Intel Core 2 Duo and 2.4GHz processor. It is, also, equipped with 4GB main memory and Windows system. In these experiments, we have taken the average of about 10 runs.

We suppose that we have Five services ($S_{i\in[1..5]}$) to be composed, and each service has three alternative services ($S_i^{j\in[1..3]}$), i.e. they have the same functionality but differ in term of their execution time (T), cost (C) and reliability (R) (cf. Table 5). The Total Cost (T.C) is the average of T, C and R.

We assume that the composite Service (CS [3]) is composed from the three services S1, S2 et S3 according to this order: CS [3] =S1 → S2 → S3. So, the composition plans of this CS [3] are shown in the Figure 9. CS[i] means that the composite service (CS) is composed from i services.

Table 5: Services alternatives with their execution time, cost and reliability

| $S_i$ | $S_1$ | | | $S_2$ | | | $S_3$ | | | $S_4$ | | | $S_5$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S_i^j$ | $S_1^1$ | $S_1^2$ | $S_1^3$ | $S_2^1$ | $S_2^2$ | $S_2^3$ | $S_3^1$ | $S_3^2$ | $S_3^3$ | $S_4^1$ | $S_4^2$ | $S_4^3$ | $S_5^1$ | $S_5^2$ | $S_5^3$ |
| **T** | 0,2 | 0,1 | 0,3 | 0,1 | 0,2 | 0,1 | 0,4 | 0,2 | 0,1 | 0,2 | 0,4 | 0,1 | 0,2 | 0,3 | 0,1 |
| **C** | 0,2 | 0,4 | 0,2 | 0,3 | 0,4 | 0,2 | 0,1 | 0,2 | 0,3 | 0,4 | 0,2 | 0,3 | 0,4 | 0,1 | 0,2 |
| **R** | 0,1 | 0,2 | 0,1 | 0,4 | 0,1 | 0,2 | 0,2 | 0,4 | 0,1 | 0,3 | 0,4 | 0,2 | 0,1 | 0,2 | 0,1 |
| **T.C** | 0,17 | 0,23 | 0,2 | 0,27 | 0,23 | 0,17 | 0,23 | 0,27 | 0,17 | 0,3 | 0,33 | 0,2 | 0,23 | 0,2 | 0,13 |

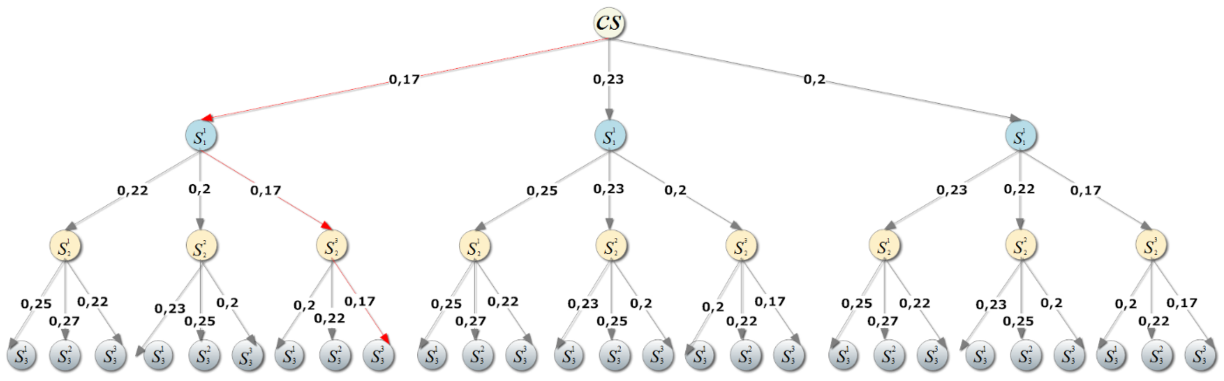


Figure 9. Composition services plan

According to Figure 9, the best plan for the composite service CS [3] is $S_1^1 \rightarrow S_2^3 \rightarrow S_3^3$. During the execution process of the composed services, the failure can occur at any moment, i.e., either $S_1^1$, $S_2^3$ or $S_3^3$ can be unavailable. To deal with this problem, we select another path by using the approach proposed in section 3. The compositions services plans, for the composites services CS [2], CS [4] and CS [5], are constructed as the same manner as the composition plan of the CS [3]. Table 6 summarizes the different composition plans for the CS [i] where i∈[2..5].

As mentioned in the above sub-section 3.3, the role of the CMA is to conceive and to create the various plans for a composite service in the form of a directed graph to meet a user demand and then, it sends the best plan to SA.

Once the SA receives the execution plan, it checks the execution of the atomic services of the composite service and then, it sends to the CMA the state of each atomic service execution, to allow it to save the recent checkpoint or to send the ideal sub-plan according to the execution state. Thus, the two agents must collaborate to provide to the end user the result of its request.

We present below the different scenarios to validate the proposed approach. In these scenarios, we note that the behavior of CMA and SA agents affects the findings explained through the three scenarios in term of the selection of the best plan or the sub-plan according to the situation (available or unavailable atomic services). However, they do not influence on the execution time (T), cost (C), reliability (R) and Total Cost (TC), because these latters depend on the selected atomic services on one hand, and the role of the CMA and SA agents is to control and to supervise the correct execution of these atomic services on the second hand.

Table 6: Best composition plans

| **Composite Service** | **Best composition plan** | **Total Cost** |
|---|---|---|
| **CS [2]** | $S_1^1 \rightarrow S_3^3$ | 0,34 |
| **CS [3]** | $S_1^1 \rightarrow S_2^3 \rightarrow S_3^3$ | 0,5 |
| **CS [4]** | $S_4^3 \rightarrow S_2^3 \rightarrow S_3^3 \rightarrow S_5^3$ | 0,67 |
| **CS [5]** | $S_4^1 \rightarrow S_2^3 \rightarrow S_3^3 \rightarrow S_5^3 \rightarrow S_1^1$ | 0,83 |

## 4.3 Scenario 1: the impact of alternative services number on the total cost

The objective of this scenario is to show the impact of alternative services number on the total cost. In this experimentation, for each composite service CS [2], CS [3], CS [4] and CS [5], we vary alternative services number from 3 to 10, and we observe the total cost for each CSi. In this simulation, we do not deal with the unavailable services. The result of this experimentation is shown in Figure 10.

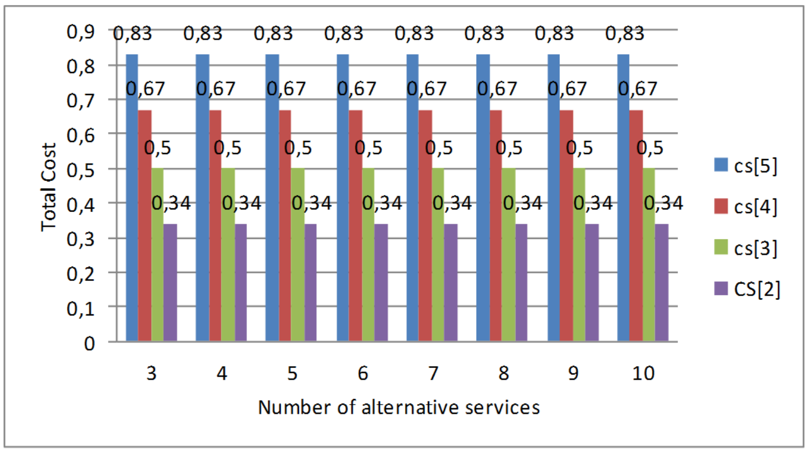


Figure 10. Impact of alternative services number on the total cost

Through this experimentation, we can note that total cost increases from 0.34 to 0.83 when the size of the composite service vary from 2 to 5. In addition, we can observe that for each composite service, its total cost is the same whatever the number of alternative services. Indeed, in the case of the services availability, our approach selects the best plan, i.e., it gives the best composition plan in term of total cost, and since all alternative services are available, the best composite service is composed by the same atomic services.

We can conclude that the total cost of the composite service depend on its size, and not on the number of alternative services. This is in the case of the services availability.

## 4.4 Scenario 2: the impact of the unavailable services on the total cost

In this experimentation, we deal with the unavailability of some alternative services. For this, we fix the number of the atomic services to three, the number of alternative services to 3 and we vary the number of unavailable services from 1 to 3. Then, we observe the impact of the unavailable services on the Total cost. In this scenario, we consider CS [3] as a composite service and, we set the three services$S_1^1$, $S_2^3$ and $S_3^3$ as unavailable services. The Table 7 summarizes the failure cases of services. Recalling that the first composition plan is the best one, and it is: CS [3]= $S_1^1 \rightarrow S_2^3 \rightarrow S_3^3$.

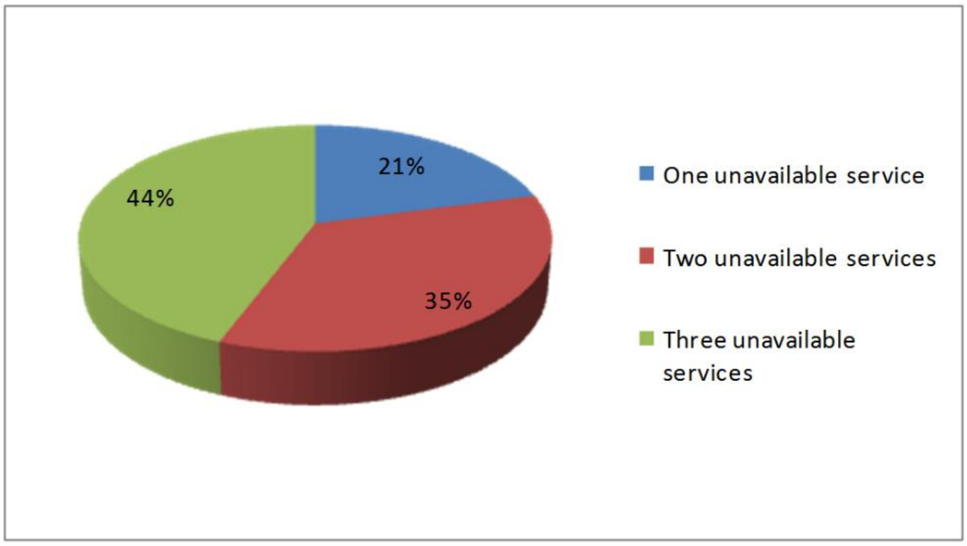


Figure 11. Percentage of the average total cost under unavailable services constraint.

As shown in the Figure 11, the percentage of the total cost increase from 21% to 44% when the number of the unavailable services increase from 1 to 3. This increase is justified by the quality of selected services that are engaged to obtain the composite service. i.e., when the number of unavailable services is one, the new plan composition is just the first one after the initial best plan, and when the number of unavailable services is three, the obtained plan composition will be the fourth plan composition in term of quality. Recalling that the total cost represent the quality of the composite service and not the response time.

## 4.5 Scenario 3: the impact of unavailable services on the response time

The goal of this scenario is to compare the execution response time (T), the cost (C) of the composite service, Reliability (R) and the Total Cost (T.C) in the case of unavailability of some services with those where all services are available. To reach this objective, we realize four experimentations. In the first one, we vary the size (the number of services that compose the needed service) of the composite service from 2 to 5, and then extract the response time for each case.

In this scenario, we compare the obtained results between the best plan (without unavailable service) and the other plans (the first, the second and the third plans). These three last plans correspond, respectively, to the case where one, two or three services of the best plan are unavailable and, thus the new composition plan is build. As in the first experimentation of this scenario, we realize two others experimentation, but the metrics to be compared concern, respectively, the cost and the total cost.

Table 7: Impact of unavailable services on the average cost

| **Number of unavailable services** | **Unavailable Services** | **New plan** | **Total Cost** | **Average Cost** |
|---|---|---|---|---|
| **1** | $S_3^3$ | $S_1^1 \rightarrow S_2^3 \rightarrow S_3^1$ | 0,53 | 0,57 |
| | $S_2^3$ | $S_1^1 \rightarrow S_2^2 \rightarrow S_3^3$ | 0,5 | |
| | $S_1^1$ | $S_1^3 \rightarrow S_2^3 \rightarrow S_3^3$ | 0,67 | |
| **2** | $S_1^1$ then $S_2^3$ | $S_1^3 \rightarrow S_2^3 \rightarrow S_3^3$ Second plan | 0,97 | 0,95 |
| | | $S_1^3 \rightarrow S_2^2 \rightarrow S_3^3$ Final plan | | |
| | $S_1^1$ then $S_3^3$ | $S_1^3 \rightarrow S_2^3 \rightarrow S_3^3$ Second plan | 0,92 | |
| | | $S_1^3 \rightarrow S_2^3 \rightarrow S_3^1$ Final plan | | |
| | $S_2^3$ then $S_3^3$ | $S_1^1 \rightarrow S_2^2 \rightarrow S_3^3$ Second plan | 0,97 | |
| | | $S_1^1 \rightarrow S_2^2 \rightarrow S_3^1$ Final plan | | |
| **3** | $S_1^1$ then $S_2^3$ and $S_3^3$ | $S_1^3 \rightarrow S_2^3 \rightarrow S_3^3$ Second plan | 1,2 | 1,2 |
| | | $S_1^3 \rightarrow S_2^2 \rightarrow S_3^3$ third plan | | |
| | | $S_1^3 \rightarrow S_2^2 \rightarrow S_3^1$ Final plan | | |

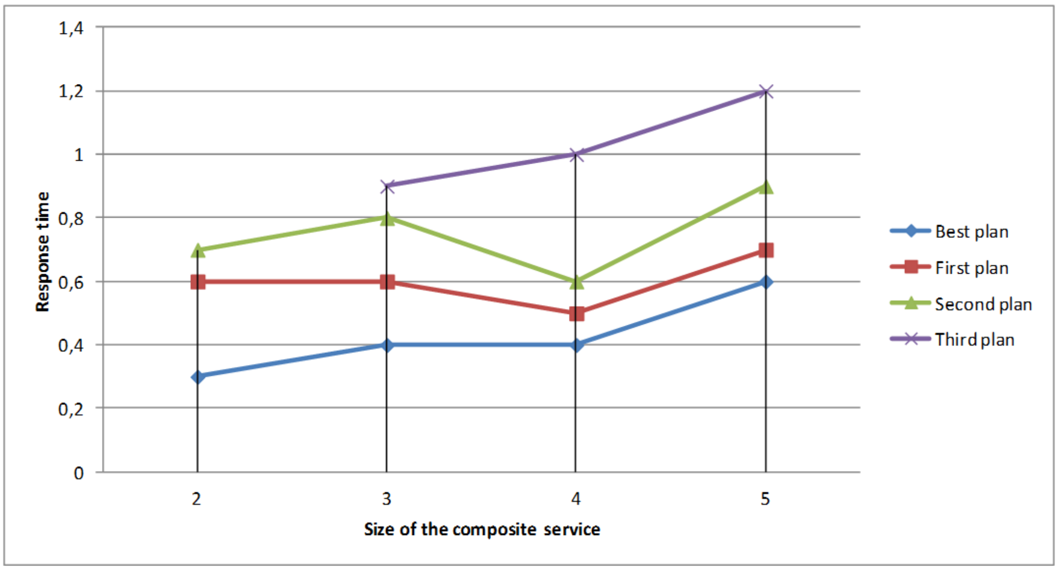


Figure 12. Impact of unavailable services on the response time

As shown in the Figure 12, the response time increases when the composite service size increases. Moreover, for each composite service size category, the response time of the best plan is better than the other plans. However, through the first and the second plans, we can note that unavailable services number has no effect on the response time. Indeed, in the case where the size of the composite services is equal to 4 (CS[4] ). The best plan, in this case, is: $S_4^3 \rightarrow S_2^3 \rightarrow S_3^3 \rightarrow S_5^3$ because according to the table 5, the Total cost of this plan is the optimal one and it is equal to 0,67. In the case where the first atomic service ($S_4^3$) that compose the CS[4] is unavailable, our approach selects the alternative of the $S_4^3$, which is the $S_4^1$. So, the first plan in this case is: $S_4^1 \rightarrow S_2^3 \rightarrow S_3^3 \rightarrow S_5^3$, its Total cost is equal to 0,77 but its response time (T) is equal to 0,5 (see table 5). In the case where the first and the second atomic services are unavailable, the obtained second plan is: $S_4^1 \rightarrow S_2^2 \rightarrow S_3^3 \rightarrow S_5^3$. And in this case, the Total cost and response time are 0,83 and 0,6 respectively

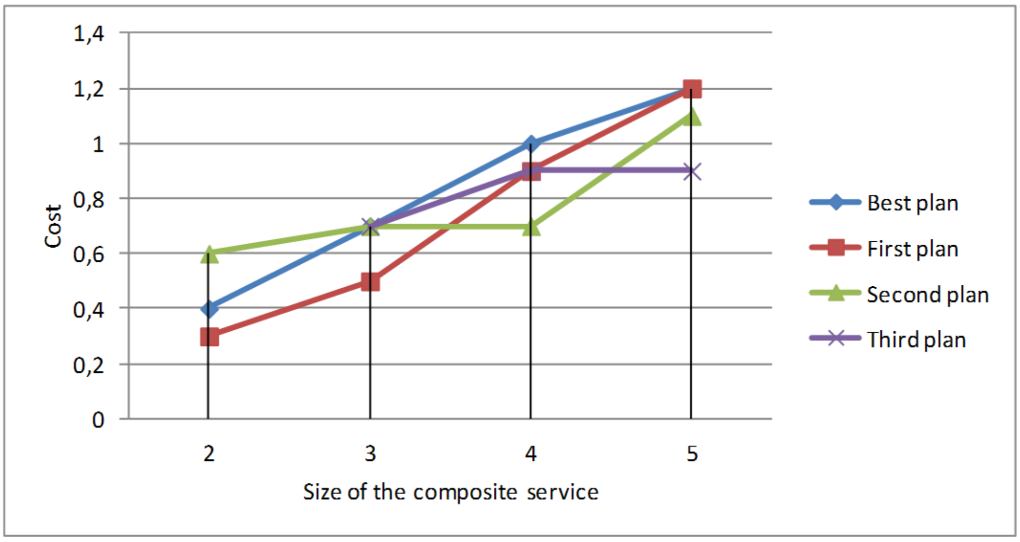


Figure 13. Impact of unavailable services on the cost.

From Figure 13, we observe that, for all plans, the cost increases when the size of the composite service increases. However, this cost does not depend on the number of unavailable services.

From both Figures 12 and 13, we can conclude that the size of the composite service has an impact on both the response time and the cost. However, the number of the unavailable services has no impact on the response time and the cost. This is due to that when an unavailable service occurs, its selected alternative may have either a response time or a cost better than the first one.

As shown in the Figure 14, the total cost, which includes the response time, the cost and the reliability of services, depend on both the size of the composite service and on the unavailable services number. Indeed, since there is a compromise between the cost and the response time, the selection of the next alternative must take into account the three metrics (Response time, cost and reliability).

Through these scenarios, we conclude that the proposed approach gives, almost, the better plan for the composite service.

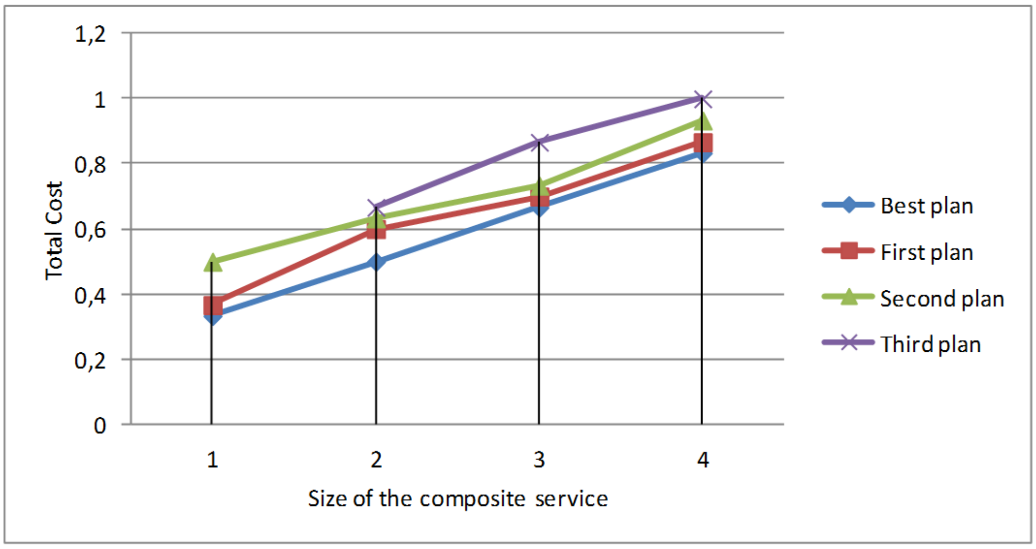


Figure 14. Impact of unavailable services on the reliability.

## 5 Conclusion

The objective of this work is to propose an approach of fault tolerance, during the execution of a composite cloud service. The latter is based on multi-agent planning, minimum necessary paths selection and backward recovery techniques. A composite cloud service is represented by a directed graph such that the nodes are atomic Cloud services. To select the minimal path, we use Dijkstra's algorithm. The latter makes it possible to find the shortest path from an initial node. The Composition Manager Agent (CMA) carries out this phase. Then, the CMA transfers the chosen minimal path to the Supervisor Agent (SA) who is responsible for proper functioning of the composite service according to this minimal path. During execution, the CMA records, each time, a service as a checkpoint provided that from this service there is at least one other sub-plan (path). If the SA detects a fault, it informs the CMA by sending a fault report, which contains at what level the error occurred and at which node. So, the CMA will return to last recorded checkpoint to choose another minimal sub-plan (which has the minimal value) by applying Dijkstra's algorithm (it takes the checkpoint as parameter), so we are going to resume execution from the last saved checkpoint and so on, until the completion of the required composite service. Finally, the CMA sends the result of the required cloud service execution to the customer.

The proposed approach is analyzed under three different scenarios. The experimental results proved that the proposed approach provides better performance based on different measures such as response time, cost and total cost of different alternatives of Cloud services. The foremost contribution of the suggested approach relies on improving the reliability of the Cloud environment.

In the future, we plan to address the system behavior when a problem occurs whereas checkpoints are not yet saved or measurement data is not yet available (first run for example). In addition, we will explain in details how the CMA will conceive and create dynamically the list of best plans according to the user's request (the composite cloud service to be executed). Also, we will consider the possibility of enriching the proposed system with decision-making agents. This aims to take into account the case when an agent (CMA or SA) suspects that it may failure, it can notify this problem to its environment in order to avoid the total blockage of the system. Additionally, we are considering using AHP (Analytic Hierarchy Process) instead of Dijkstra algorithm because it could presents better results in term of finding the best plans. Finally, we aim to evaluate the proposed approach in more detail through more complicated case studies of real systems.

## References

[1] P. Keerthika, P. Suresh, R. Manjula Devi, M. Sangeetha, and C. Sagana, “Design of a fault tolerant strategy for resource scheduling in cloud environment,” *Int. J. Eng. Adv. Technol.*, vol. 9, no. 1, pp. 5121–5128, 2019, doi: 10.35940/ijeat.A1519.109119.

[2] K. Saidi, O. Hioual, and A. Siam, “Novel energy-aware approach to resource allocation in cloud computing,” *Multiagent Grid Syst.*, vol. 17, no. 3, pp. 197–218, 2021.

[3] A. Rezaeipanah, M. Mojarad, and A. Fakhari, “Providing a new approach to increase fault tolerance in cloud computing using fuzzy logic,” *Int. J. Comput. Appl.*, pp. 1–9, 2020.

[4] A. Mimouni, A. Agoune, and O. Hioual, “Back Recovery Protocol Based Multi-Agent Planning for the Fault Tolerance of Composite Cloud services,” in *12th edition of the Conference on Advances of Decisional Systems. ASD 2018: Big data & Applications.*, 2018, pp. 138–150.

[5] O. Hioual, Z. Boufaïda, and S. M. Hemam, “Load balancing, cost and response time minimisation issues in agent-based multi cloud service composition,” *Int. J. Internet Protoc. Technol.*, vol. 10, no. 2, pp. 73–88, 2017, doi: 10.1504/IJIPT.2017.085187.

[6] M. Amoon, “Adaptive Framework for Reliable Cloud Computing Environment,” *IEEE Access*, vol. 4, pp. 9469–9478, 2016, doi: 10.1109/ACCESS.2016.2623633.

[7] V. Sathiyamoorthi, P. Keerthika, P. Suresh, Z. Zhang, A. P. Rao, and K. Logeswaran, “Adaptive fault tolerant resource allocation scheme for cloud computing environments,” *J. Organ. End User Comput.*, vol. 33, no. 5, pp. 1–24, 2021, doi: 10.4018/JOEUC.20210901.oa7.

[8] A. Bala and I. Chana, “Fault tolerance-challenges, techniques and implementation in cloud computing,” *Int. J. Comput. Sci. Issues*, vol. 9, no. 1, p. 288, 2012.

[9] S. Prathiba and S. Sowvarnica, “Survey of failures and fault tolerance in cloud,” in *2017 2nd International Conference on Computing and Communications Technologies (ICCCT)*, 2017, pp. 169–172.

[10] G. Vallee *et al.*, “A framework for proactive fault tolerance,” in *2008 Third International Conference on Availability, Reliability and Security*, 2008, pp. 659–664.

[11] K. M. Sim, “Agent-based approaches for intelligent intercloud resource allocation,” *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 442–455, 2016.

[12] G. Kalanirnika and R. V. M. Sivagami, “Fault Tolerance in Cloud Using Reactive and Proactive Techniques,” *Int. J. Comput. Sci. Eng. Commun.*, vol. 3, no. 3, pp. 1159–1164, 2015.

[13] F. Wang, X. Liu, and Y. Yang, “Necessary and sufficient checkpoint selection for temporal verification of high-confidence cloud workflow systems,” *Sci. China Inf. Sci.*, vol. 58, no. 5, pp. 1–16, 2015.

[14] K. Zhang, J. Xu, M. R. Min, G. Jiang, K. Pelechrinis, and H. Zhang, “Automated IT system failure prediction: A deep learning approach,” in *2016 IEEE International Conference on Big Data (Big Data)*, 2016, pp. 1291–1300.

[15] R. Santhosh and T. Ravichandran, “Non-Preemptive Real Time Scheduling using Checkpointing Algorithm for Cloud Computing,” *Int. J. Comput. Appl.*, vol. 80, no. 9, 2013.

[16] P. Zhang, S. Shu, and M. Zhou, “An online fault detection model and strategies based on SVM-grid in clouds,” *IEEE/CAA J. Autom. Sin.*, vol. 5, no. 2, pp. 445–456, 2018, doi: 10.1109/JAS.2017.7510817.

[17] S. Di, Y. Robert, F. Vivien, D. Kondo, C.-L. Wang, and F. Cappello, “Optimization of cloud task processing with checkpoint-restart mechanism,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[18] A. Ragmani, A. Elomri, N. Abghour, K. Moussaid, M. Rida, and E. Badidi, “Adaptive fault-tolerant model for improving cloud computing performance using artificial neural network,” *Procedia Comput. Sci.*, vol. 170, pp. 929–934, 2020, doi: 10.1016/j.procs.2020.03.106.

[19] A. Ragmani, A. Elomri, N. Abghour, K. Moussaid, and M. Rida, “FACO: A hybrid fuzzy ant colony optimization algorithm for virtual machine scheduling in high-performance cloud computing,” *J. Ambient Intell. Humaniz. Comput.*, pp. 1–13, 2019.

[20] T. Wang, J. Xu, W. Zhang, Z. Gu, and H. Zhong, “Self-adaptive cloud monitoring with online anomaly detection,” *Futur. Gener. Comput. Syst.*, vol. 80, pp. 89–101, 2018.

[21] K. T. Bui, L. Van Vo, C. M. Nguyen, T. V. Pham, and H. C. Tran, “A fault detection and diagnosis approach for multi-tier application in cloud computing,” *J. Commun. Networks*, vol. 22, no. 5, pp. 399–414, 2020, doi: 10.1109/JCN.2020.000023.

[22] D. Talia, “Cloud Computing and Software Agents: Towards Cloud Intelligent Services.,” in *WOA*, 2011, vol. 11, pp. 2–6.

[23] F. D. la Prieta and J. M. Corchado, “Cloud computing and multiagent systems, a promising relationship,” in *Intelligent agents in data-intensive computing*, Springer, 2016, pp. 143–161.

[24] R. Gopinadh and K. Saravanan, “Cloud service reservation using PTN mechanism in ontology enhanced agent-based system,” *Int. J. Eng. Trends Technol.*, vol. 4, pp. 791–798, 2013.

[25] A. Singh, D. Juneja, and M. Malhotra, “A novel agent based autonomous and service composition framework for cost optimization of resource provisioning in cloud computing,” *J. King Saud Univ. Inf. Sci.*, vol. 29, no. 1, pp. 19–28, 2017.

[26] A. Belgacem, S. Mahmoudi, and M. Kihl, “Intelligent multi-agent reinforcement learning model for resources allocation in cloud computing,” *J. King Saud Univ. Inf. Sci.*, 2022.

[27] E. W. Dijkstra, *A short introduction to the art of programming*, vol. 4. Technische Hogeschool Eindhoven Eindhoven, 1971.

[28] C. Kathpal and R. Garg, “Survey on Fault-Tolerance-Aware Scheduling in Cloud Computing,” in *Information and Communication Technology for Competitive Strategies*, Springer, 2019, pp. 275–283.

[29] B. . Coghlan and J. . Jones, “Stable Memory Operations,” *Irish Pat. Appl. 1094/91 Deriv.*, 1991.

[30] B. . Coghlan and J. O. Jones, “Memory Checkpointing,” *tent Appl. 2784/92 Deriv.*, 1992.

[31] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Softw. Pract. Exp.*, vol. 41, no. 1, pp. 23–50, 2011.

[32] F. Bellifemine, A. Poggi, and G. Rimassa, “JADE–A FIPA-compliant agent framework,” in *Proceedings of PAAM*, 1999, vol. 99, no. 97–108, p. 33.