Cloud Services Orchestration: A Comparative Study of Existing Approaches

Khadija Bousselmi  
Faculty of Sciences of Tunis  
Tunis, Tunisia  
Email: khadija.bousselmi@gmail.com

Zaki Brahmi  
ISITCOM, University of Sousse  
Sousse, Tunisia  
Email: zakibrahmi@yahoo.fr

Mohamed Mohsen Gammoudi  
ISAMM, University of Mannouba  
Mannouba, Tunisia  
Email: gammoudimomo@gmail.com

Abstract—Cloud Computing is emerging today as a service model used to relocate locally-based data and applications to virtualized services available via Internet at a lower cost. A key to exploit the benefits of this model is orchestration which consists in coordinating effectively the deployment of a set of virtualized services in order to fulfill operational and quality objectives of end users and Cloud providers. Cloud orchestration can be carried out at two levels: hardware level orchestration and software level orchestration. In this paper, we highlight the main challenging points about the Cloud orchestration concept. Then, we carry on a comparative study of some existing research works involved with this concept at hardware and software levels.

Keywords—Comparative study, Cloud Computing, orchestration, Infrastructure as a service, Software as a Service.

I. INTRODUCTION

The Cloud Computing paradigm is shifting computing from physical hardware and locally managed software-enabled platforms to virtualized cloud-hosted services [1]. Cloud Computing brings together all types of virtualized services namely physical resources (CPU, server, network ...) and software resources (databases, monitoring systems, management systems, data ...). It follows a service-oriented business model [2]. This model can be organized in three layers: a first layer of services representative for material resources (Infrastructure As a Service or IaaS), a second layer comprising development platforms and deployment services (Platform As a Service or PaaS) and a third layer representing software services (Software as a Service or SaaS).

Key to exploiting the potential of Cloud computing is the issue of Resource orchestration [1]. With Cloud orchestration we have the ability to control and arrange a set of underlying technology infrastructure (hardware and hypervisor) and match them with the intended commands inputted by the users to create a set of automated events that delivers the request with the maximum efficiency [7]. The user request can be expressed via a business process or using the Cloud resources specifications. Therefore, the overall objective of the orchestration is to ensure the effective deployment of all its services in order to meet user objectives.

Like some recent studies [1][2][3], we argue that the orchestration of services in the Cloud environment is a fastidious task, and this because of the scalability, resources heterogeneity and other constraints related to the environment itself. These constraints result from a number of factors such as the limited capacity of resources (e.g., bandwidth and memory), failures (e.g., failure of a network link), the user access model (e.g., the number of users and their locations) [3] and heterogeneity of Cloud providers (the access model to resources and their configuration are specific for each Cloud provider) [14].

The main contributions of this paper are: (a) presenting the main motivating issues about the Cloud orchestration concept. (b) Comparing the main existing research works involved with the Cloud orchestration issue. (c) Highlighting the main challenging points from the comparison of these works.

The remainder of the paper is organized as follows: Section 2 introduces the Cloud services orchestration concept and its main motivating issues. Section 3 offers an overview of some existing works on the Cloud orchestration issue. Section 4 presents a comparison of studied approaches and Section 5 concludes the paper and outlines areas for future research.

II. ORCHESTRATION AND CHALLENGES

Cloud orchestration is a new buzzword in the industry; it was defined differently by researchers according to the context of its implementation. Indeed, the orchestration concept has been widely studied in the context of Web services [21] and it was extended to handle other domains like Cloud Computing. In this domain, orchestration involves different types of services (computation, storage and network) and it can be realized at different layers of the Cloud services model. We can distinguish two types of Cloud orchestration approaches: orchestration of software services (or SaaSO) and orchestration of hardware services (or IaaSO).

A. SaaSO challenges

SaaS services, usually implemented as Web services, can be orchestrated as an executable business process that interacts with other internal and external SaaS services. Interactions occur at the message level. They include business logic and task execution order; and they can span applications and organizations to define a transactional model of multi-step process in the long term [20]. Since Cloud software services are associated with material resources on which they will run, SaaSO includes the organization, coordination and management of automated deployment and configuration of one or more interrelated components necessary to the provision of services at a given time. SaaSO process involves two aspects: resource allocation and resource coordination [9]. For the first aspect, the overall objective is to guarantee service-level
agreements (SLAs) of Cloud resources to make cost-efficient decisions for workload placement. The problem of placing orchestration workloads onto Cloud physical servers shares similar characteristics with an existing problem called Component Placement Problem (CPP) [15]. CPP can be divided into two categories: offline CPP and online CPP. In online CPP, the placement of the components is made during the runtime [15]. The main goal of online CPP is to optimize the resource usage by the components, and at the same time to minimize the overall response time. This implies that a solution of service placement should ensure resources provisioning by optimizing the use of involved Cloud resources. It must also be able to generate statistics on the state of the available resources in order to optimize their use. For these reasons, a successful placement solution, to our understanding, should verify these requirements:

- **Scheduling algorithm**: the scheduler is a key component of a placement solution which is charged to generate the execution plan of the orchestration. The latter results from calculating all possible task-resource mappings and selecting the best ones according to the orchestration objectives. Since orchestrations are usually represented as directed acyclic graphs (DAG), scheduling, in DAG structures, is still NP-complete and computable in polynomial time only for some simple cases [12]. Besides, the number of tasks of an orchestration as well as the number of available resources can be huge in the Cloud environment. Thus, the time required to compute an execution plan depends mostly on the used scheduling algorithm (heuristic, evolutionary).

- **Infrastructure description**: Cloud providers, typically, have different geographically distributed data centers. Each of these data centers might offer different resource types (virtual machines, servers, storage, network). These resources could be classified over three categories, according to the goal they were designed for: computing (CPU, virtual machines), storage (services, databases), network (routers, switches) as described in [5][9]. Each resource has its own properties (state, capacity), operations (how to allocate, start, stop) and configuration method. Consequently, modeling orchestration resources is a primary step for placement solutions.

- **Provider heterogeneity**: remember that the goal of a resource orchestrator is to select, assemble, deploy, and manage a set of software units (appliances) delivering particular application functionality. There is currently no widely accepted standard for appliance virtualization format, which means that an appliance is built for a specific virtualization technology and possibly will not run on Cloud resources that are managed by non-overlapping virtualization technologies [1]. Heterogeneity in Multi-Clouds requires that resources across different Cloud infrastructures are expressed in a uniform, platform-independent manner [15].

- **Provisioning algorithm**: Since the common objectives for placement solutions include minimizing the resource usage while maintaining the applications performances, considering the provision of all orchestration resource types placement is primordial. For instance, at platform level, most existing works are focusing on the management of VM mapping to physical servers, while at the application level, the plan is to manage VM resources based on the applications workload [15] which lacks integration of storage and network resources, such as servers, connectivity, bandwidth and latency.

The second aspect of the orchestration process (resource coordination) implies optimal deployment of allocated resources in order to achieve the users operational requirements (such as minimizing the total cost or response time of the orchestration). These requirements are either functional insuring the success of execution of different Cloud services or non-functional consisting of verifying parameters of service level agreement (SLA) of the orchestrations services. SLA verification includes assessing the quality of service (QoS) metrics such as service reliability, latency, availability,... For this reason, the orchestration solution must insure real time monitoring of involved services during its deployment and envisage alternative execution plans if any error occurs. Another aspect that should be taken into account during execution is inter-dependencies between orchestration services. Also, the orchestration deployment should be aware of the deployment method used for each service since it will directly impact the performance of the overall orchestration. In recent literature, although few works have been conducted to address the challenge of service deployment across multiple data centers, most of them have not taken the service dependencies into consideration, where services are independently deployed to optimize the corresponding performance and operational cost [16].

**B. IaaS challenges**

IaaS involves the creation, management and manipulation of Cloud resources (i.e., computation, storage and network) in order to realize user requests in a Cloud environment, or to realize operational objectives of the Cloud service provider. User requests are driven by the service abstraction and service logic that the Cloud environment exposes to them. Examples of operational objectives that require orchestration functions include decreasing costs by consolidating physical resources and improving the ability to fulfill service level agreements (SLAs) by dynamically reallocating compute cycles and network bandwidth [5]. Cloud infrastructure orchestration has been handled in the context of Cloud management systems [9]. These systems have to verify the following requirements:

- **Visibility**: the management system should monitor the status of Cloud resources to ensure their availability to end users and providers. They should, also, provide detailed information about available resources such as their visibility status, cost, placement and other information that could be useful for the system users. Visibility status could be private (visible only for their tenants) or public (visible for all system users).
• **Orchestration:** as in SaaS, IaaS involves two issues: resource allocation and resource coordination. For the first issue, IaaS needs to guarantee service-level agreements (SLAs) of datacenter resources, such as network bandwidth and latency for tenants [9]. The second issue ensures the effective configuration and execution of involved resources. Several techniques have been proposed to ensure IaaS such as using common databases techniques [5].

• **Provisioning:** to ensure effective provisioning of Cloud resources, both tenants and datacenter operators should provide statistics about resources utilization and techniques to optimize their use such as auto-scaling solutions. They have also to handle resource volatility (change of resource status without user intervention) and provide necessary actions to ensure system consistency.

## III. OVERVIEW OF EXISTING SOLUTIONS

Orchestration is a challenging issue for many domains. In the Cloud Computing context, several approaches have been proposed to handle services orchestration [12][13][15][16]. These approaches could be classified onto two categories according to the orchestrated service’s type: SaaS orchestration approaches (SaaSO) and IaaS orchestration approaches (IAASO). The first category encloses approaches concerned with the orchestration of virtualized software services, including provisioning of hardware resources on which these services will be executed, while the second category focuses on the orchestration and provisioning of Cloud hardware resources at the infrastructural level.

### A. SaaSO approaches

They are designed for coordination and effective implementation of several services together in a single process such as in [11][12][15][16][19]. We will expand the description of studied solutions according to the considered aspect of orchestration: resource allocation or resource coordination.

### Resource Allocation

In [11], authors have proposed a new platform for the dynamic selection of the orchestration services location during execution runtime. The proposed solution consists in adapting BPEL based compositions to the Cloud environment where the selection of service location can be processed at runtime. Moreover, as current BPEL implementations do not provide mechanisms to schedule service calls with respect to the load of the target hosts, the proposed platform creates automatically new instances of unavailable or overloaded services as well as needed and assigns them to their endpoints. To determine whether a service is overloaded, the authors of the approach use a department supervisor to determine the service load in real time based on the following ratio: the sum of the loads assigned to the service on the overall CPU capacity of the service. If this ratio exceeds a definite threshold, the service is considered overloaded and cannot respond to new requests. In [12], the authors of [11] proposed an extension of their platform, named CADAS (Cost And Data flow-Aware Scheduler), in order to respond to some of its limitations, mainly the target of the scheduling strategy. The user can express his orchestration objectives in terms of a weighted value of the two criteria (response time and cost) by assigning a weight to each criterion. The scheduling algorithm is charged of determining if it would be better to wait for the availability of a requested resource (in case of overload) to perform a service or if it should transfer data to a remote similar service. To do this, the description model of the Cloud resources has been expanded to include information about the network connection between different remote areas (or data centers), which is a criterion that can affect both the cost and the execution time of the service orchestration. While the minimization of orchestrations response time has been already handled in [11], the decision of the scheduling algorithm to wait until a specific resource is available again was handled in [12]. The execution of the remaining dependent services of the orchestration is suspended at their turn. This was insured by taking into account the dependencies between the different resources during the execution of the services orchestration. The new proposed scheduling algorithm is a genetic algorithm that uses collected heuristics about all orchestration tasks to guarantee the best scheduling time.

The authors in [15], at their turn, signaled that none of researches on CPP has considered the placement of the applications’ data together with the applications’ components in the context of Cloud service composition. To address this problem, they proposed a new approach to re-architecting Cloud data centers for better management of SaaS compositions, especially improving service performance by minimizing its response time on one side and optimizing the use of Cloud resources by provisioning resources and their data on the other side. The approach focuses on the composition level and does not involve the deployment phase of the service composition.

In [15], the authors focus on the initial allocation of hardware resources to software components and the optimization of the location of these resources before composition execution. To do this, the authors formalize the two problems as combinatorial optimization problems and use the evolutionary algorithms, particularly genetic algorithms, to solve them. Based on the experimental results, the proposed algorithms always produce a feasible and satisfactory solution to all the test problems, although, the computation time taken still quite long. In addition to this, the interdependencies between the compositions services have not been addressed by the proposed solution which considers each component independent from the other one.

### Resource coordination

Few researches have granted attention to the issue of Multi-Cloud orchestration deployment [16][18][19]. The authors of [16] are particularly interested in the issue of Cloud services selection for the deployment of SOA applications that takes into account the interdependencies between services. The proposed solution consists of a generic platform for the deployment of service-oriented applications in a distributed Cloud environment. The general architecture of the proposed solution consists of two levels: a deployment manager for SOA applications at each data center and a local scheduler at each Cloud service (a VM or a physical
server). The deployment manager is used to analyze the interdependencies between services and predict the number of user queries in order to determine the optimal locations of the Cloud services to be allocated using a genetic algorithm, whereas the local scheduler is used to determine and dynamically generate the number of instances of the invoked services that can respond to all users submitted queries. Conductor [18][19] is another platform proposed to handle the challenges related to the orchestration deployment of computations in the Cloud such as heterogeneity of service providers, dynamic pricing of service tasks, errors and exceptions management and the interdependencies between storage and computing services. Conductor platform insures better decisions about selecting Cloud services based on the optimization goals of the user (minimizing the cost or the execution time) by determining the optimal execution plan that meets these goals using a solver and deploying it. Conductor also allows supervising the execution of the orchestration and detecting deviations according to the predetermined execution plan such as the degradation of a specific service performance or its unavailability. If an anomaly is detected, the system recalculates the execution plan to determine the new optimal one and adjusts the deployment of the orchestration under the new plan. To perform the various tasks of the system, Conductor proceeds by three steps. It starts by the creation of the initial system model. The system models the computing objectives of the user (operations and data exchange) and his optimization goals concerning cost and execution time, then models computing and storage services. Once the system is modeled, the approach move to determining the optimal execution plan by selecting Cloud services that better meet the optimization goals of the user using a solver. Then, the system deploys the predetermined execution plan and monitors services in order to detect anomalies in real-time (performance degradation, unavailability...).

B. IaaS approaches

Few works in the field of Cloud orchestration have focused on IaaS services. These works are presented as Cloud resources management systems and propose scalable platforms for Cloud resource orchestration such as in [4] [5] [8] [9]. These solutions formulate the requirements of the orchestration using physical resources specifications. They also help to maintain the consistency of Cloud resources [4]. Some of these approaches are designed to meet a specific need [8], others offer generic solutions to define several orchestration scenarios for different types of Cloud resources (computing resources, storage and network) [4] [5] [9]. Polyphony [6] is a specific Workflow orchestration framework for Cloud Computing. It aims to exploit the computing power offered by Cloud resources to expedite the processing of images from Mars and improve their processing and delivery time. Polyphony is implemented using Eclipse Equinox, a popular implementation of the OSGi (Open Services Gateway Initiative) specification. The proposed approach integrates a distributed queue called SQS (offered by Amazon EC2 [10]), used to publish and distribute tasks to nodes that can perform them. Each node has a software application that allows it to interact with the queue. One of the objectives of the Polyphony framework is to exploit the idle cycles of machines to perform tasks in progress and accelerate the processing performed by different nodes.

The authors of [5] proposed a framework, named Data-centric Management Framework, for hardware services orchestration. This solution aims to fulfill Cloud resources orchestration requirements such as concurrent access to resources, resource heterogeneity and resource integrity. Taking into account the fact that such issues have been addressed through well-known database solutions, the proposed solution integrated databases management capabilities, namely the control of concurrency, the application of integrity constraints and atomic transactions in order to realize orchestration operations which insures ACID (Atomicity, Consistency, Integrity and Durability) properties of performed operations.

The overall architecture of the proposed solution is composed of two layers: physical layer and logical layer. At the first layer, DMF contains a conceptually centralized repository of all managed resources (computing, storage and network) and a physical view of these resources such as the physical model and actions. This level allows interacting directly with Cloud resources to execute user queries. The higher level includes the logical data model, the user views, actions and procedures carried out by users. This solution consists of three main elements: a single formal data model, a declarative language for data manipulation and a transactional semantics. The adopted data model is a hierarchical model in which resources are organized in a tree structure composed of three main groups namely: storage resources, computing resources and network resources. For each resource, there are two copies that represent its state: the primary copy is stored at the physical layer and the secondary copy is at the logical layer. A user can specify views and integrity constraints in a declarative query language. The proposed language is domain-specific that supports the following main elements: the views, constraints, actions and procedures as in database systems.

The proposed framework, however, suffers from some limitations. There is no consideration of volatile resources which is contradictory to the dynamic nature of Cloud resources that could change states frequently. Besides, the proposed approach is centered on a single data center and could not be scalable in the case of multi-cloud systems. Based on these limits, the authors of [8] proposed some extensions to the DMF framework to meet requirements concerning, in particular, scalability, availability, security and the management of access competition to resources. In fact, the new framework, called TROPIC, provides robust hardware services. Robustness means that failures in an orchestration transaction do not lead to an undefined behavior or inconsistent states of the system. On the one hand, the separation of logical and physical level ensures the consistency of the system by avoiding illegal operations. On the other hand, the system architecture is designated to deal with the volatility of the Cloud environment whenever a system inconsistency is detected at the physical level by performing necessary correction operations. Besides, the safety of Cloud services is ensured by checking integrity constraints of each orchestration transaction. Moreover, the management of concurrent access to Cloud resources when running multiple transactions simultaneously is insured by
implementing a pessimistic concurrency control algorithm. Finally, the comparison of the TROPIC platform with Amazon EC2 platform [10] shows that the first one can restore faster from system failures. COPE (Cloud Orchestration Policy Engine) [4], is another extension of the DMF framework aiming to automate the process of Cloud resources orchestration based on the service operators objectives. Indeed, the Cloud resource management APIs are generally handled in a specific different way and do not allow to automate the orchestration process or to express more sophisticated orchestration goals. To address this problem, the COPE platform, based on the goals and constraints of Cloud providers, automates the process of Cloud resources orchestration using a declarative domain-specific language. COPE allows, also, Cloud providers to optimize their orchestration decisions via a solver that considers the orchestration transaction as a constraint optimization problem in order to generate optimal orchestration decisions. COPE platform can operate in two modes (centralized and distributed). In centralized mode, there is only one COPE instance that manages all hardware resources of the Cloud. In distributed mode, there are several instances of COPE nodes, one for each Cloud data center which allows management of multi-provider Cloud resources. Lastly, LiveCloud [9] is a recently proposed Cloud resource management framework. It aims to fill in some limitations on the existing Cloud orchestration solutions. In particular, the existing approaches, generally, do not consider the impact of the choice of network devices on the orchestration (bandwidth, network topology, network latency ...). Besides, existing approaches assume that system users know the status and properties of the various system resources, but because of the volatility of the Cloud environment, the system visibility remains limited. To do this, LiveCloud approach offers high data visibility using three different views:

- The physical data view represents the topology of the data center showing various existing resources and their properties.
- The logical data view that allows to define in an abstract way the different resources of the system as a set of numeric attributes (CPU frequency, number of cores ...) and all other metrics calculated automatically (rental period, SLA parameters ...). It is used, also, to define Cloud resource orchestrations.
- The service provider view which is specific to each service provider. It allows him to define his own network topology and available resources as well as the properties of each one.

IV. COMPARISON OF STUDIED APPROACHES

In this section, we present a comparative study of the studied approaches. This comparison can be divided into two parts: comparison of SaaS orchestration approaches and comparison of IaaS orchestration approaches.

A. Comparison of SaaS approaches

We begin by evaluating each studied approach according to the requirements identified in section II-A, including adopted solutions for orchestration scheduling, provisioning and deployment.

The approach proposed in [11] has the major advantage of keeping the running time of orchestrations almost constant regardless of the number of allocated resources. However, it was centralized over one type of resources (virtual machines) and lacks inclusion of other Cloud resources such as storage and network ones. Besides, the proposed approach doesn’t take into account inter-dependencies between the orchestration services or the data flow transmission over executed services. The proposed provisioning algorithm is based only on the service load; however, other criteria may influence the performance of a Cloud service such as its overall quality of service (QoS) or cost. In addition to this, the scheduling strategy target is only minimizing execution time of the orchestration regardless of its cost which can increase significantly if many services are overloaded [12].

The extension of the [11] approach in CADAS platform, allowed taking into account the users non-functional requirements in the deployment strategy, namely, cost and response time. Nevertheless, the proposed approach suffers from some shortcomings, particularly: The criteria expressing the user’s preferences are limited to the cost and execution time; however, an end user can involve other criteria in the choice of Cloud services such as its quality of services (QoS) and the service level agreement parameters (SLA). Besides, the proposed approach is limited only to the specific resource provider Amazon EC2 and does not address the case of multi-tenant Cloud services that may require different methods of deployment and specific access models [16].

The platform proposed in [15], has the advantage of considering storage resources in the management of Cloud orchestrations. It also used genetic algorithms to solve problems of initial allocation of hardware resources to software components and the optimization of the location of these resources before composition execution. The proposed algorithms have as objective to minimize cost and response time of the orchestration at the same time. Yet, the deployment strategy hasn’t been considered in [15].

In [16], the authors provided a deployment strategy for SOA applications using distributed Cloud services which allows minimizing the overall response time of the application. However, the proposed deployment strategy does not take into account other constraints that may affect the performance of the application such as the maximum number of simultaneous requests allowed for a single service, the physical constraints related to the service instance (bandwidth, storage capacity,) and those of the hosting data center (like its storage capacity) in addition to the heterogeneity of the specifications of the Cloud resources for different suppliers. The latter can prevent, in some cases, the communication between application components and can cause the failure of execution of the overall application. Besides, the proposed solution considers that the execution time of a given Cloud service is constant while, in reality, it’s proportional to the task assigned to the service and can vary from one application to another.

The Conductor approach [19] provided acceptable results, regarding cost and execution time, in comparison with other...
deployment test scenarios without having to store the data locally. The proposed approach presents some limitations. It is limited to the deployment of computations (virtual machines) and its only target is to minimize the overall response time of the orchestration. Moreover, the system is based on a static evaluation of the Cloud services performance (predicted execution time) which increments the probability of execution plan deviations during runtime. Since Cloud environment is very dynamic, the solution adopted by the Conductor approach can cause a significant loss of time due to the recalculation of the execution plan every time an anomaly is detected.

As a summary, table I shows the comparison of the studied SaaS orchestration approaches. Note that "-" symbol, in both tables I and II, indicates that given requirement has not been handled in the corresponding solution. The studied approaches used either Linear Programs or Genetic Algorithms for scheduling, provisioning and deployment of Cloud orchestrations. Both methods tend to determine a way to achieve the best outcome using a set of inequalities or constraints. Linear programs, such in [11] [19], deal with a set of input data controlled via a number of objective functions; however, a Genetic Algorithm, such in [12] [15] [16], uses a fitness function to select data from randomly generated generations of data from an initial selected one. Genetic Algorithms seem more suitable for handling issues of services orchestration in a Cloud environment because of their dynamicity, moreover, proposed solutions using linear programs are often costly and can be unbounded.

Regarding orchestration scheduling, most studied solutions are specific to one type of Cloud resources (virtual machines), while storage resource have been taken into account only in approaches [15] and [19]. For Cloud resources provisioning, all approaches tend to improve orchestrations response time and/or cost which represents the main SLA parameters that interest an end user. But, other SLA parameters can influence the performance of the orchestration as exception handling capacity, network properties (bandwidth, latency). Concerning deployment techniques, only [16] and [19] approaches propose solutions adapted to the deployment of distributed Cloud services. The approach in [16] takes into account the communication time between data centers and remote Cloud services to generate the execution plan of the orchestration. However, this solution is not appropriate in the case of deploying large scale orchestrations [17]. The approach in [19], designed for heavy computation ends, allows detecting deviations in the execution plan at runtime and readjusting it by recalculating the new optimal plan.

B. Comparison of IaaS approaches

As in the previous section, we begin by evaluating each studied approach according to the requirements enumerated in section II-A, in particular, system visibility, orchestration method and provisioning strategy. Polyphony [6] is a specific platform designed for the end of rapid deployment of services orchestrations. It uses a commercial solution (Amazon EC2) for effective orchestration and provisioning of Cloud resources, but it doesn’t consider resource volatility during orchestration execution. The TROPIC [8] and DMF [5] platforms, despite their benefits, suffer from some limitations, especially, volatile Cloud resources aren’t handled. Moreover, a problem of scalability can arise when the user wants to execute a transaction orchestration involving a large number of resources of the same data center. In addition to this, orchestration scenarios are limited to a single VM migration scenario; however, the needs of orchestrations of end users and service operators can be much more sophisticated and involve several types of Cloud resources at the same time. Similarly, we note some limitations for the COPE approach [4]. It lacks management of volatile resources and interdependencies between different resources when executing an orchestration. The same problem of scalability as in DMF approach can arise when the user wants to run a large orchestration transaction involving services of the same data center. Furthermore, the study of orchestration policies based on non-functional user objectives (cost optimization, memory capacity optimization) has not been taken into account since both service orchestration scenarios tested are based on functional objectives of service operators. However, a COPE user can express his goals using SLA parameters, assessed statically at the initial stage. Provisioning of managed resources is insured by regular verification of their availability and the overall system consistency as in database solutions. The main advantage of LiveCloud platform [9] is insurance of resource visibility for all its users but it suffers from lacking consideration of volatile Cloud resources. The managed resources properties, likewise, are based on static measures of SLA parameters values, whereas these parameters could change due to the dynamic nature of the Cloud environment and therefore they should be monitored regularly.

To sum up, table II shows the comparison of studied IaaS orchestration solutions. From this table, we can deduce that resources visibility is insured for all users only in Polyphony and LiveCloud approaches, the other solutions suppose that resources can be viewed or accessed only by their providers. Besides, the provisioning of resources has been provided only by the two approaches COPE and Polyphony. The first optimizes resource usage by considering the orchestration as a constraint optimization problem. The second approach optimizes resources usage using Amazon EC2 auto-scaling solution. None of the aforementioned approaches has considered volatile resources in its orchestration solution. Also, most studied solutions used domain-specific transactional languages to perform orchestration operations. This choice allows insuring Cloud resources consistency but should be learned by end users to manipulate it. Moreover, verification of SLA parameters was performed only in COPE and LiveCloud approaches and is limited to the use of static values extracted from providers SLA contracts without verification or update.

V. Conclusion and Future Work

Cloud computing is a recent technology that allows to relocate data and applications on a dematerialized infrastructure accessible from the Internet as virtualized services at a lower cost. A key concept that allows highlighting the potential of Cloud computing environment is orchestration that aims to coordinate the execution of a set of virtualized services within the same process. Only a few recent papers attempted to study the concept of Cloud orchestration. The proposed approaches focused on
either SaaS services orchestration as in [11][12][15][19] or on IaaS services orchestration as in [4][5][8][9]. Although these works offer custom solutions for Cloud orchestration scheduling, provisioning and deployment, no approach has met both the management of orchestrated software services (their scheduling, deployment and services interdependencies between them) and the effective provisioning of implied Cloud resources (their placement, configuration, migration and interdependencies between them) at the same time to ensure the effective implementation of the orchestration and meet the quality requirements of the end user. Furthermore, they did not consider all resource types (computing, storage, or network), nor the volatility of these resources at runtime.

For future work we intend to propose a new approach for Cloud services orchestration that guarantees the satisfaction of services’ SLA parameters and end users objectives. Our target solution should also insure the provisioning of Cloud resources. To this end, we will investigate the possibility of extending COPE approach to take into account SaaS services.

### REFERENCES


---

**TABLE I: SaaS approaches comparison**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm type</td>
<td>Linear program</td>
<td>Genetic Algorithm</td>
<td>Genetic Algorithm</td>
<td>Dynamic Linear program</td>
</tr>
<tr>
<td>Resource type</td>
<td>Virtual Machines</td>
<td>Virtual Machines</td>
<td>Virtual Machines</td>
<td>Virtual Machines</td>
</tr>
<tr>
<td>Provisioning</td>
<td>Provider Heterogeneity</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Objective</td>
<td>Minimizing response time</td>
<td>Minimizing response time</td>
<td>Minimizing response time</td>
<td>Minimizing response time</td>
</tr>
<tr>
<td>Algorithm Type</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Genetic Algorithm</td>
</tr>
</tbody>
</table>

**TABLE II: IaaS approaches comparison**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>System type</td>
<td>Specific</td>
<td>Genetic</td>
<td>Genetic</td>
<td>Generic</td>
<td>Generic</td>
</tr>
<tr>
<td>Resource Visibility</td>
<td>Public</td>
<td>Private</td>
<td>Private</td>
<td>Private</td>
<td>Public</td>
</tr>
<tr>
<td>Orchestration technique</td>
<td>Simple Service EC2</td>
<td>Queuing (Amazon)</td>
<td>Specific Transactional Language</td>
<td>Specific Transactional Language</td>
<td>Specific Transactional Language</td>
</tr>
<tr>
<td>Provisioning Technique</td>
<td>Simple Service EC2</td>
<td>Queuing (Amazon)</td>
<td>-</td>
<td>Database techniques</td>
<td>-</td>
</tr>
<tr>
<td>Volatility Management</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Monitored SLA parameters</td>
<td>Response time</td>
<td>-</td>
<td>-</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>

---

7


