Agent-based cloud workflow execution

J. Octavio Gutierrez-Garcia and Kwang Mong Sim

Department of Information and Communications, Gwangju Institute of Science and Technology, Buk-gu, Gwangju, Republic of Korea

Abstract. Cloud computing offers an economical and feasible solution for scientific workflow applications requiring large amounts of computational resources and expensive hardware. Supporting Cloud workflow execution involves: (i) allocating and composing a collection of Cloud resources, and (ii) coordinating distributed and self-interested participants. The contributions of this research are: (i) proposing an agent-based approach for supporting workflow execution in one or multiple Clouds, (ii) defining Petri-net based methodologies to design workflows and Cloud resources that sustain concurrent and parallel management of workflows, (iii) implementing an agent-based testbed to simulate distributed workflow execution, and (iv) providing empirical evidence to demonstrate the effectiveness and efficiency of agent-based Cloud workflow execution. The agents are endowed with distributed algorithms, e.g., contract net protocol, to allocate and compose Cloud resources based on workflow requirements. Simulation results demonstrated that: (i) Agents effectively executed (with a 100% success rate) workflows autonomously, even when dealing with concurrent workflow executions, (ii) task parallelization was efficiently achieved in randomly created workflows with different levels of parallelism and ordering constraints, (iii) workflow execution was efficiently achieved since the makespan and number of messages exchanged increased linearly with the number of tasks.

Keywords: Agent-based Cloud computing, Cloud workflow execution, resource allocation, multi-agent systems

1. Introduction

A workflow is a series of interrelated steps that model the execution of a process. Scientific workflow applications [14] cover a broad variety of domains: bioinformatics [24], astronomy [12], geology [31], etc. Scientific workflows often demand high amounts of computational resources, in addition to many specialized services to execute a large number of resource-consuming tasks. This results in high investments in hardware infrastructure that may only be needed for executing workflows on one occasion. Cloud computing is a collection of distributed and remotely accessible computing resources, which should be dynamically virtualized, composed, and provisioned based on consumers’ needs and subject to service level agreements established through negotiation [7]. Clouds emerge as a feasible and economical solution to scientific workflow applications, where resources are offered in a pay-as-you-go model. However, executing workflows in the Cloud frequently requires dynamically allocating and composing distributed and heterogeneous services that may be provided by different Cloud providers located in multiple Clouds.

Cloud services deployed as web service components are normally partial solutions that must be dynamically coordinated to execute a Cloud workflow (CW). Furthermore, Cloud workflow execution must support both dynamic resource allocation and dynamic service composition in response to workflow tasks’ required services as well as dealing with distributed, self-interested, and autonomous parties such as Cloud providers, brokers, and Cloud consumers. In addition, these parties should interact and coordinate among themselves to execute CWs in highly parallel and concurrent environments. All this accentuates the need for an agent-based solution [4,8,21,23]. Agents are autonomous problem solvers that can act flexibly, e.g., interacting with other agents through negotiation (see [28]) and cooperation (see [25]). In addition, agent-based solutions have...
proved to be efficient in a wide range of dynamic environments, from human tracking video surveillance systems [15], interactive artistic installations [9], energy management [26], shipboard systems [34], oceanic monitoring [5] to establishing service-level agreements in Cloud-computing environments [27].

Whereas workflow management systems such as: GridBus [33], Pegasus [16], SwinDeW-G [32], etc., support workflow execution in distributed and heterogeneous environments, further adaptation is needed to facilitate multiple Cloud environments. In addition, current workflow management systems lack cost-based service selection mechanisms. Moreover, most of them include a planning phase, where workflow tasks are matched with Cloud services. In doing so, complete knowledge of the environment is needed to obtain workflow execution plans and since Clouds are distributed by nature, complete knowledge is a strong assumption.

In this paper, an agent-based Cloud workflow execution approach supported by dynamic resource allocation and composition is proposed. Cloud participants: Cloud consumers, Brokers, Cloud providers, and Cloud resources are represented by agents. Agents are endowed with reactive distributed problem solving techniques that allow Cloud workflow execution in a responsive manner based on currently deployed Cloud services without planning ahead. The well-known contract net protocol (CNP) [30] is used as a service selection mechanism based on service fees. In addition, agents are capable of handling concurrent and parallel workflow executions. Moreover, formal methodologies for defining both CWs and services’ workflows that handle synchronization and coordination aspects are defined through the use of colored Petri nets [13] and nested Petri nets [19]. Both Petri net formalisms are combined to support concurrent and parallel Cloud workflow execution through agent collaboration.

The contributions of this work are as follows:

1) Devising and implementing an agent-based testbed that supports distributed Cloud workflow execution (Section 3).

2) Designing agent behaviors (Section 4) to efficiently allocate and compose Cloud services in a concurrent and parallel manner.

3) Integrating a cost-based service selection mechanism into the Cloud workflow management system (Sections 4.2 and 4.3).

4) Defining formal methodologies to model both Cloud resources and CWs based on colored and nested Petri nets (Sections 4.4 and 4.5).

5) Providing empirical evidence (Section 5) to demonstrate the effectiveness and efficiency of the proposed agent-based workflow execution approach.

In addition, Section 2 presents a review of related work and Section 6 includes a discussion. Finally, Section 7 summarizes the contributions of this work relative to other related works and describes future research directions.

2. Related work

Resource allocation methods can be divided into centralized and distributed mechanisms. Whereas centralized mechanisms [3,20] contain a single entity that oversees services and has the authority to allocate appropriate services to tasks, resource allocation in distributed mechanisms results from the interaction among the parties involved. In this domain, auction mechanisms have been widely adopted [17,18], where either consumers or service providers bid for services and jobs, respectively. Moreover, agent-based automated bargaining methods [29] have been used to allocate services and to negotiate service terms among the interested parties.

In the area of Cloud service composition, the research effort presented in [36] addressed service composition in multi-Cloud environments as a combinatorial optimization problem by creating a search tree from the services deployed on the Clouds. Finally, artificial intelligence planning techniques are applied to the tree to compose atomic services based on a set of requirements. In [35], a semantic matching algorithm that uses web services’ descriptions to match the inputs and outputs of correlated web services is presented. In [10], a combination of acquaintance networks and CNP is integrated into an agent-based Cloud service composition approach. Acquaintance networks are used to deal with incomplete knowledge about the Cloud environment while the CNP is used for establishing contracts for Cloud services.

Current workflow management systems such as: GridBus [33], Pegasus [16], and SwinDeW-G [32], have been adapted from Grid to Cloud-computing environments.

Pegasus [16] has a planning module to match tasks to services. The matching relation is then handed to the execution engine that executes tasks according to their ordering constraints. The planning module is a cen-
3. **Agent-based Cloud workflow execution architecture**

The multi-agent system architecture that supports Cloud workflow execution is presented in Fig. 1. The components of the testbed are as follows:

A service ontology is a formal representation of services and of the atomic requirements that the services can fulfill. The definitions of the atomic requirements included in the service ontology are provided by the system designer.

A directory is a listing of available service provider. Each entry in the directory associated to a service provider contains its name, location (e.g., URI address), and capabilities, which are expressed as a list of requirements that the service provider can satisfy.

A semantic web service is a web service whose definition is mapped to an ontology, which characterizes the capabilities of the service as well as its application domain. The ontological definition of the web services is based on the service ontology, so a one-to-one match-
ing between workflow tasks’ requirements and Cloud service requirements can take place.

A resource agent (RA) orchestrates a semantic web service. In addition, an RA is a service wrapper that controls and provides a standard access to the semantic web service, which gives access to a Cloud resource. Furthermore, an RA contains a reference to an atomic requirement, which can be fulfilled by its semantic web service.

A service provider agent (SPA) handles a set of RAs that may be published and located in several Clouds. In this regard, a service provider can be seen as the owner of a virtual organization, where some RAs are registered as members. The functions of SPAs are: (i) (de)allocating, coordinating, and synchronizing RAs, (ii) providing both a common context and the rules that direct the achievement of a shared objective, (iii) keeping a record of all the RAs linked to it (each entry of this record contains the location of an RA and the requirement that fulfills), and (iv) publishing its description in the directory.

A broker agent (BA) manages the execution of CWs requested by consumer agents. To execute CWs, BAs select and compose services deployed on Clouds.

A BA utilizes requirements included in the CW to search for service providers published in the directory; it then starts the composition process by adopting the CNP [30] for selecting services from available SPAs and then executing workflow tasks.

The CNP consists of two roles: initiator (manager) and participant (contractor). The initiator agent (e.g., a consumer agent) sends a call for proposals to execute a job (e.g., a CW) to n participant agents (e.g., BAs). The participant agents may send: (i) refuse messages to dismiss the call for proposals or (ii) propose messages to bid for the job. Afterwards, the initiator agent evaluates the k received proposals and sends: (i) an accept-proposal message to the participant agent of the winner bid, and (ii) k − 1 reject-proposal messages to the remaining participants. Finally, the selected participant agent executes the job and sends the result to the initiator agent.

In addition, as a result of the process for satisfying requirements, SPAs may need additional requirements, e.g., a computing service provider may need a storage address to store its results. These requirements are handed to the BA, which again enacts the CNP for selecting complementary service providers. Further details are presented in Section 4.

A consumer agent (CA) acts on behalf of a consumer to interact with BAs and its functions are: (i) mapping workflow tasks to registered Cloud requirements in the service ontology by means of one-to-one matching between tasks’ requirements and available Cloud service requirements, and (ii) contracting the best (cheapest) BA to execute a CW. This is achieved by means of adopting the initiator role of the CNP. CAs submit CWs to BAs as soon as CAs are provided with CWs by Cloud consumers.

The agent-based Cloud workflow execution architecture (Fig. 1) is topologically distributed, thus CAs, BAs, and SPAs can be located in independent remote servers, another server can contain the service ontology and the directory, and RAs can be located in remote Cloud resources accessed via internet.

4. Agent models for Cloud workflow execution

4.1. Petri nets background

In colored Petri nets, tokens have colors that are data types. Places only contain tokens of a particular color, and both transitions and arcs may have logical expressions that prevent transitions from occurring unless the expressions are evaluated as true.

Nested Petri nets [19] may contain n Petri net levels, where each Petri-net level may contain Petri nets as tokens, which may also contain inner-level Petri nets as tokens, and so on. In addition, nested Petri net may contain colors and uncolored tokens. The evolution of nested Petri nets is captured in two types of transitions: (i) Local transitions remove, add, and create tokens in the level where the transitions are located. (ii) Synchronized transitions extend local transitions by adding binding constraints between objects nets (inner-level nets) and system nets (outer-level nets). Figure 2 shows a system net with an initial marking comprised of an object net ON1 in place p11. Synchronized transition st1 \↓ of the system net has an inner-level (↑) binding constraint (i.e., a synchronization with object nets) to enable the triggering. Object net ON1 contains a synchronized transition st1 ↑ that has an outer-level (↓) binding constraint (i.e., a synchronization with the system net) to enable the triggering. In Fig. 2, synchronized transition st1 is enabled in both the system net and the object net, because the binding constraints are met and the tokens required by both transitions are present.

Nested Petri nets were adopted for providing system designers with a formal tool to devise and implement CWs and RAs capable of running in both a parallel and
The workflow execution approach is modeled using two levels of nested Petri nets: the high level to model agent behavior (Figs 3 and 4), and the low level to model RAs (Fig. 5) and CWs (Fig. 6).

4.2. Broker agent model: Workflow executor

The model of a BA (Fig. 3) represents the system net that coordinates the concurrent execution of CW object nets. The model of BAs is based on both: the participant and the initiator role of the CNP. Whereas each CA plays the role of an initiator, each SPA plays the role of a participant.

Since the BA model contains CW object nets, inner-level synchronization with the CW object nets is required. The synchronization is achieved by means of synchronized transitions $s_{t1}$, $s_{t2}$, $s_{t3}$, and $s_{t4}$, which are shared between the BA model and the CW object net (Fig. 6). Table 1 contains detailed descriptions of BAs’ actions and the associated events represented by the transitions of the BA model.

A firing sequence $\sigma = \{t_1, t_2, s_{t1}, t_3, s_{t3}, t_4, t_5\}$ represents the simplest coordination process among one CA, one BA, and $n$ SPAs, where the BA receives a call for proposals to execute a CW ($t_1$) from a CA. Then, the BA starts the execution of the CW ($t_2$). Consequently, the BA sends a call for proposals to fulfill a requirement ($s_{t1}$) to $n$ SPAs by adopting the initiator role of the CNP to contract an SPA. Then, the BA receives the output ($t_3$) of the requirement from the SPA and evolves the CW ($s_{t3}$). For explanation purposes, it is assumed that the CW contains only one requirement. Then, the workflow is set ready to be delivered ($s_{t4}$) and the output is sent ($t_5$) to the CA. This firing sequence is assumed to be accompanied by inner-level firing sequences corresponding to the CW object net.
4.3. Service provider agent model: Service manager

The model of an SPA (Fig. 4) represents the system net that coordinates independent and evolving RA object nets. Since the SPA model contains RA object nets, inner-level synchronization with the RA object nets is required. The synchronization is achieved by means of synchronized transitions $s_{t5}$, $s_{t6}$, $s_{t7}$, $s_{t8}$, and $s_{t9}$, which are shared between the SPA model and the RA object net (see Fig. 5). Table 2 contains detailed descriptions of SPAs’ actions and the associated events represented by the transitions of the SPA model.

A firing sequence $\sigma_a = \{t_1^a, s_{t5}^a, s_{t6}^a, t_2^a\}$ represents the simplest coordination process for an SPA that contains $n$ RAs and $k$ previously assigned requirements by BAs. Transition $t_1$ activates the coordination process by receiving $k$ accept messages from BAs to fulfill requirements. Then, transition $s_{t5}$ assigns the new requirements to available RAs. Transition $s_{t6}$ receives the resultant outputs from the RAs, and finally, transition $t_2$ delivers the output to the BAs. This firing sequence is assumed to be accompanied by RAs’ firing sequences such as $\sigma_b = \{s_{t5}^b, t_1^b, s_{t9}^b\}$, which corresponds to the model of the RA object net contained in Fig. 4. The resultant interleaving firing sequence is $\sigma = \{\sigma_a, t_1^a, \sigma_a, s_{t5}^a, \sigma_a, s_{t6}^a, \sigma_b, t_1^b, \sigma_b, s_{t6}^b, \sigma_a, s_{t9}^a, \sigma_a, s_{t9}^b\}$.

4.4. Resource agent model

RAs (Fig. 5) are object nets, i.e., inner Petri nets contained in a system net, in this case, an SPA model. The
Table 1

Description of transitions for the broker agent model

<table>
<thead>
<tr>
<th>Id</th>
<th>BAs’ actions and events represented by the transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>Handle a CA’s call for proposals by adopting the CNP</td>
</tr>
<tr>
<td>t₂</td>
<td>Start the execution of the CW</td>
</tr>
<tr>
<td>t₃</td>
<td>Receive tasks’ outputs from previously contracted SPAs</td>
</tr>
<tr>
<td>t₄</td>
<td>Receive a request from an SPA regarding a set of needed requirements to fulfill a previously assigned requirement (as explained in Section 3)</td>
</tr>
<tr>
<td>t₅</td>
<td>Send an inform message to either a CA or an SPA containing CWs’ outputs.</td>
</tr>
<tr>
<td>s₁</td>
<td>Send a call for proposals (adopting the CNP) to SPAs to contract services to fulfill a given requirement</td>
</tr>
<tr>
<td>s₂</td>
<td>Repeat the CNP in case of error during workflow execution</td>
</tr>
<tr>
<td>s₃</td>
<td>Update the progress of the CW</td>
</tr>
<tr>
<td>s₄</td>
<td>Prepare workflow outputs to be delivered</td>
</tr>
</tbody>
</table>

Fig. 6. Cloud workflow model patterns.

The main structure of the RA model has two places and two synchronized transitions st₅ and st₆ (see Fig. 5). Transition st₅ synchronizes the beginning of the RA’s workflow with the reception of a request message from an SPA. This transition has a condition attached, denoted by if [Req = X₁], in order to be triggered. The condition simply consists of accepting requirements that can be handled by the web service in question. Transition st₆ reports the output to the outer-level Petri net (an SPA).

In addition to the main structure, the RA model has two design patterns (Fig. 5). Design pattern I allows an RA to ask for internal requirements (st₈) and wait until this requirement is resolved by another RA (st₉) belonging to the same SPA. This gives rise to the definition of internal requirements, which can be resolved in the same virtual organization, i.e., the requirement can be resolved by another RA object net included in the service provider system net. Pattern II has a similar structure, but it differs in the first synchronized transition, which is used to ask for external requirements (st₇), i.e., these requirements cannot be resolved by any existing RA object net of the current service provider system net. In this case, the SPA sends the requirement to a BA, which has to search for another service provider who can fulfill the requirement. Both patterns I and II share the same synchronized transition for receiving results st₉ and proceeding with the internal RA’s workflow.

4.5. Cloud workflow model

CWs are object nets contained in the systems nets of CAs, BAs, and SPAs. However, CWs are mainly managed by BAs. CWs are executed by BAs, which are in charge of selecting and composing Cloud services to execute all workflow tasks. Service composition is
Table 2
Description of transitions for the service provider agent model

<table>
<thead>
<tr>
<th>Id</th>
<th>SPAs’ actions and events represented by the transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Receive an accept message indicating that the SPA has won a bid to execute a workflow task as a result of a previous CNP execution with BAs.</td>
</tr>
<tr>
<td>t2</td>
<td>Send workflow tasks’ outputs to BAs</td>
</tr>
<tr>
<td>t3</td>
<td>Receive outputs from BAs regarding previous requests</td>
</tr>
<tr>
<td>t4</td>
<td>Send requirements to BAs (as explained in Section 3)</td>
</tr>
<tr>
<td>s5</td>
<td>Assign unfulfilled requirements by allocating available RAs</td>
</tr>
<tr>
<td>s6</td>
<td>Deallocate an RA by receiving its resolved requirement. This resets the RA in order to accept another requirement</td>
</tr>
<tr>
<td>s7</td>
<td>Receive request messages from RAs asking for external requirements (i.e., requirements that cannot be resolved by any existing RA object net of the current SPA system net)</td>
</tr>
<tr>
<td>s8</td>
<td>Receive request messages from RAs asking for internal requirements (i.e., requirements that can be resolved by another RA object net included in the SPA system net)</td>
</tr>
<tr>
<td>s9</td>
<td>Deliver recently arrived and fulfilled requirements to the corresponding RAs.</td>
</tr>
</tbody>
</table>

Directed by the requirements of workflow tasks. A CW provides ordering constraints to service compositions. In addition, parallelism and concurrency are supported by the CWs’ underlying structure.

The main structure of CWs contains (Fig. 6a) two transitions: $t_1$ and $s_4$.

Transition $t_1$ starts the execution of the CW. Input: one uncolored token from the initial place $p_a$. Output: $n$ (input) tokens to every place $p_i$ that contains no initial ordering constraints.

Transition $s_4$ delivers the workflow output. Input: $m$ (output) tokens from every place $p_k$ that contains a final output of the CW. Output: $\emptyset$. Synchronization details: Outer-level synchronization with transition $s_4$ of the BA Petri net model.

Task patterns (Fig. 6b) are used as building blocks to create complex CWs (see example presented in Fig. 7). A task pattern contains three places: (i) a place $p_b$ that contains the initial (input/output) of previous tasks, (ii) a place $p_c$ that contains the task to be executed and the service needed to execute it (Task, Req), which is expressed by a requirement, and (iii) a place $p_d$ that receives the output from executing the task. Task patterns contain only one transition:

Transition $t$ represents the handing over of tasks. Input: one (input) or (output) token from place $p_b$, and one (Task, Req) token from place $p_c$. Output: one (Task, Req) token in place $p_e$ of the synchronization module (Fig. 6c).

The synchronization module of a CW (Fig. 6c) synchronizes the execution of the CW with the system net, in this case the model of BAs. The function of each synchronized transition is as follows:

Transition $s_1$ synchronizes the submission of workflow tasks with the transmission of a call for proposals from a BA to SPAs. Input: one (Task, Req) token from place $p_c$ that contains all enabled workflow tasks. Output: one (Task, Req) token in place $p_e$ of the synchronization module (Fig. 6c) as a record for the execution in progress of the workflow task in $p_f$. Synchronization details: Outer-level synchronization with transition $s_1$ of the BA system net.
Sections 3 and 4. The testbed was implemented using Java and the JADE agent framework [6].

The computer specifications on which the experiments were carried out are as follows: Intel Core 2 Duo E8500, 3.16 GHz clock speed, 4 GB RAM, Windows Vista Enterprise (32 bits), service pack 2.

5.1. Evaluating the management of CWs of different sizes

**Objective**

A series of experiments was conducted to evaluate the proposed agent-based Cloud workflow execution approach when handling CWs of different sizes composed of homogeneous and heterogeneous tasks executed in homogeneous and heterogeneous Cloud-computing environments.

**Experimental settings**

Tables 3 and 4 show the input data and parameters of the testbed. Table 3 shows the parameters regarding the structure of CWs: (i) number of tasks, (ii) number of instructions for both homogeneous and heterogeneous tasks, (iii) required service types for executing both homogeneous and heterogeneous tasks, (iv) number of workflow segments, i.e., parallelism level, and (v) connectivity level, i.e., number of ordering constraints. Table 4 shows the parameters regarding the agent-based testbed: (i) number of agents, (ii) workflow request rate, (iii) number of service types, (iv) Cloud service types and their processing capacities, and (v) number of agents involved per CNP enactment.

CWs were created randomly based on two parameters: connectivity level and workflow segments. The number of workflow segments indicates the number of sections where workflow tasks are positioned, for example, a CW that contains 20 tasks with 5 workflow segments means that tasks are arranged in 5 sections, each containing 4 tasks. The connectivity level indicates the number of ordering constraints among workflow tasks. This is defined by the number of outgoing transitions.

Transition $s_{12}$ allow resubmitting workflow tasks, when it is necessary to repeat the CNP either because in the BA system net there were no proposals or because the fulfillment of the requirement failed. Input: one (Task, Req) token from place $p_f$. Output: one (Task, Req) token in place $p_e$. Synchronization details: Outer-level synchronization with transition $s_{12}$ of the BA Petri net model.

Transition $s_{13}$ represents the execution of a workflow task, once the required service is allocated by the BA. Input: one (Task, Req) token from place $p_f$. Output: one (output) token in place $p_g$ of the corresponding task pattern that has as a precondition the execution of the task in question. This is determined according to the conditions attached to the output arcs, denoted by (task.ID = i). Synchronization details: Outer-level synchronization with transition $s_{12}$ of the BA Petri net model.

Task patterns are joined by means of task connector patterns (Fig. 6d), which consist of: Transition $t_x$ enables $r$ workflow tasks. Input: $q$ (output) tokens from $q$ places $p_d$, which indicate $q$ preconditions, and one uncolored token from place $p_c$ to trigger task synchronization and close down previously executed tasks. Output: $r$ (output) tokens in $r$ places $p_b$ to enable $r$ workflow tasks.

5. Evaluation

A group of experiments were carried out using the agent-based testbed and Petri net models defined in
arcs from tasks to forward tasks, i.e., tasks located in forward segments, e.g., a task with a connectivity level of 3 located in the first segment can have up to 3 arcs towards tasks located in forward segments. In addition, the selection of tasks to connect with was determined randomly. Nevertheless the randomness was based on the right half of a normal distribution to guarantee that closer segments were more connected than more distant segments, under the assumption that closer tasks have more dependency among them. Figure 8 shows two CWs that contain 20 tasks each, with 5 workflow segments and connectivity levels of 1 and 3, respectively. The CW shown in Fig. 8a has a similar structure as the real-world workflow presented in [24], while the CW shown in Fig. 8b represents a CW endowed with complex ordering constraints. It should be noted that although both CWs are 5-segment, not all the tasks are perfectly aligned by their corresponding segments, i.e., workflow segments may have different number of tasks among them (see Fig. 8), due to the randomly creation of ordering constraints.

The CWs included in the experiments contained 5 workflow segments with a connectivity level of 5 to execute generic randomly created CWs with an average parallelism and connectivity.

The number of tasks varied from 10 to 100 to take into account a broad variety of workflows’ sizes.

Experiments were conducted with CWs that contained only homogeneous tasks and with CWs that contained only heterogeneous tasks.

In homogeneous CWs, two different sizes of homogeneous tasks were used: 1-instruction and 1000-instruction tasks. This was to evaluate the time that agents takes to achieve the workflow execution without spending a considerable amount of time executing tasks (1-instruction tasks). Then, from that basis, the time that agents take to execute regular homogeneous tasks (1000-instruction tasks) is measured. To simulate homogeneous task executions, only one required service type to execute tasks was used: $s_5$, to focus the evaluation on measuring the capabilities of agents to handle concurrent workflow execution of different sizes.

CWs with heterogeneous tasks allowed a full evaluation of the synchronization and concurrency capabilities of the agents involved in the workflow execution. The required service type to execute tasks was assigned randomly among 10 types, which can be mapped to Amazon EC2 instances [1]. In heterogeneous CWs, both required service type and number of instructions per task were randomly assigned.

To simulate execution time, Cloud services were designed to wait for a number of seconds before generating an output. The execution time of a task was determined by dividing its corresponding number of instructions by the processing capacity of the Cloud service where the task was executed. Then, in java code, the execution time of a task $t$ on a service $s$ was simulated as follows: $Thread.sleep(t.getNoInsts()/s.getProcessingCapacity())$.

The number of agents involved was fixed to 10 CAs, 10 BAs, 10 SPAs and 1000 RAs, to concentrate the evaluation on workflow execution rather than on inter-layer agent communication. RAs were randomly allocated among the 10 SPAs.

The workflow request rate indicates the number of workflow execution request per second. Agents may implement web services to provide access to Cloud services and some commercial web services [2] support a request rate of one request per second. To facilitate its proper functioning for web-based deployments, the agents were designed to handle execution request rates higher than that of web services.

The services implemented by RAs were allocated randomly from the available service types to generate heterogeneous Cloud providers. In experiments involving homogeneous tasks, only one service was available: $s_5$, and in experiments involving heterogeneous tasks, the 10 service types were available.
The number of agents involved per CNP enactment was limited to a maximum of 4 (1 initiator agent and 3 participant agents). This limit allows a proper sample of prices for executing a task, following the structure of the price study presented in [22], and prevents unnecessary exchange of messages.

In addition, whenever a participant agent either BA or SPA received a call-for-proposal, the content of their proposals, i.e., prices for their services, was randomly determined to avoid concentrating the execution of CWs in a few BAs and SPAs and to promote a highly distributed workflow execution.

A total of 3 configurations of the agent-based testbed were used to evaluate the proposed agent-based Cloud workflow execution approach: (i) 1-instruction workflow tasks and homogeneous services, (ii) 1000-instruction workflow tasks and homogeneous services, and (iii) heterogeneous workflow tasks and heterogeneous services.

For each configuration of the testbed, 10 experiment runs varying the number of tasks were carried out, each consisting of 10 concurrent executions, for an overall of 300 Cloud workflow executions.

Performance measures

The performance measures are: (i) Percentage of successful workflow executions, (ii) average workflow makespan (amount of time required to complete a workflow), and (iii) average number of messages exchanged. See Table 5 for details.

Simulation results are shown in Figs 9 and 10. From these results, five observations were drawn.

Observation 1

Given randomly created CWs of different sizes, containing homogeneous and heterogeneous tasks executed by homogeneous or heterogeneous services, the agents achieved a 100% success rate in executing CWs in a concurrent manner.

Analysis

The agents achieved a 100% success rate in executing CWs. By following the CNP, all the agents in all the workflow executions were capable of contracting proper SPAs that executed either homogeneous or heterogeneous workflow tasks.

In the CNP, CAs could contract BAs to carry out the CWs. In addition, BAs could select different SPAs with appropriate resources to execute tasks and both BAs and SPAs could make proposals for executing both CWs and workflow tasks, respectively.

Table 5

<table>
<thead>
<tr>
<th>Performance measures for experiments 5.1 and 5.2</th>
<th>( \frac{N_S}{N_A} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentage of successful workflow executions</td>
<td>( \sum(WM_S)N_S )</td>
</tr>
<tr>
<td>Average workflow makespan</td>
<td>( \sum(M_A)N_A )</td>
</tr>
<tr>
<td>Average number of messages</td>
<td>( N_S ): No. of successful workflow executions</td>
</tr>
<tr>
<td></td>
<td>( N_A ): No. of attempted workflow executions</td>
</tr>
<tr>
<td>WM_S: Workflow makespan</td>
<td>M_A: No. of messages for attempted workflow executions</td>
</tr>
</tbody>
</table>

Despite executing CWs in a concurrent manner, no deadlock situations (e.g., allocating the same Cloud service for two different BAs) occurred during the concurrent execution of the CWs. This is because agent behaviors (i.e., agents’ functionalities) of SPAs in charge of allocating and releasing Cloud services were implemented as synchronized methods that prevented interleaved method invocations, which may have resulted in deadlocks.

In summary, the agents achieved a 100% success rate in executing randomly generated CWs because through dynamic and flexibly selection, CAs had a high chance in finding BAs that could select and combine a group of SPAs with matching resource capabilities and prices for executing the CWs.

Observation 2

In general, the average makespan increased as the size of CWs increased. In addition, CWs containing homogeneous tasks showed a makespan following an ascending linear-like pattern.

Analysis

As shown in Fig. 9, as the number of tasks increased, the makespan increased for the three cases: both 1-instruction and 1000-instruction tasks that required homogeneous services, and heterogeneous tasks that required heterogeneous services. This is because with increasingly more tasks to be executed, increasingly more services needed to be contracted. This shows that the makespan increased linearly with the size of CWs.

Observation 3

In general, the average makespan of CWs containing homogeneous tasks executed by homogeneous services showed less variation with respect to the number of workflow tasks than CWs containing heterogeneous tasks executed by heterogeneous services.

Analysis

As shown in Fig. 9, the execution of CWs with homogeneous tasks kept a steady and smooth relation with respect to the average makespan. This is because the
processing capacities of Cloud services and amount of instructions per tasks were homogeneous, i.e., all the tasks took the same amount of time to be executed. Then, the slight variations were due to the additional agent interaction needed to assign the additional workflow tasks among the SPAs.

As shown in Fig. 9, an irregular pattern of the average makespan was recorded when CWs contained heterogeneous tasks executed by heterogeneous services. This is because both the required services and the amount of instructions were assigned to tasks randomly, for instance, a task containing 1800 million of instructions could have been executed by a slow service, e.g., $s_2$, with a processing capacity of 400 million of instructions per second. Then, the high variations were due to the time needed to execute the workflow tasks by RAs.

This result shows that agents promptly reacted to the completion of tasks, regardless of either the amount of instructions per tasks or whether the required services were homogeneous or heterogeneous.
Observation 4

In general, the average makespan of CWs containing 1-instruction and 1000-instruction homogeneous tasks showed a similar ascending pattern with a 5-second bias in all size categories.

Analysis

As shown in Fig. 9, the makespan in CWs containing 1-instruction and 1000-instruction homogeneous tasks had almost the same pattern with respect to the number of tasks. The execution of CWs containing 1-instruction tasks (see Fig. 9) shows the time (except for the 1 ms that took to execute each task) that resulted from the accumulated message latencies and the time that the agents took to coordinate the distributed execution of the CWs, i.e., the accumulated time that workflow tasks waited in order to be executed. Each executed CW had 5 segments, thus the CWs contained interleaved paths of (on average) 5-task long from the start node to the final node. Then, when 1000-instruction workflow tasks were executed by services with a processing capacity of 1000 million of instructions per second, which processed tasks in 1 s, an overall of 5 s were needed to execute the CWs. This caused the 5-second bias between 1-instruction and 1000-instruction CWs.

Since CWs were composed of 5-task long paths and each 1000-instruction task took 1 s to be executed, the 5-second bias between the 1-instruction and 1000-instruction tasks shows that agents were capable of parallelizing the execution of tasks. For example, all the initial tasks of all the 5-task long paths included into the CWs were executed in parallel, since there were no ordering constraints, then the executed tasks habituated the execution of the tasks contained in the second segment of the CWs, and so on, similarly to how multiple independent pipelines work. For each task execution, BAs adopted multiple CNP with SPAs in parallel, which also allocated Cloud resources in parallel for each task.

This result shows that the agents can efficiently parallelize the execution of workflow tasks by adopting multiple CNP in parallel.

Observation 5

The number of messages exchanged showed a linear-like pattern with respect to the number of tasks contained in the CWs.

Analysis

As shown in Fig. 10, the number of messages exchanged showed a constant rate and ascending pattern with respect to the number of workflow tasks. The number of messages exchanged was due mainly to selecting services by means of adopting the CNP with feasible SPAs, i.e., SPAs that can fulfill a given requirement. Thus, the more tasks there were in a CW, the more required services needed to be contracted using the CNP, resulting in a direct relation between the number of workflow tasks and the number of messages exchanged.

5.2. Evaluating the management of CWs of different parallelism and connectivity levels

Objective

A series of experiments was conducted to: (i) Evaluate the effectiveness and efficiency of the agent-based Cloud workflow execution approach in accomplishing the execution of concurrent CWs with different parallelism and connectivity levels. (ii) Evaluate the capabilities of agents to parallelize the execution of CWs when possible.

Experimental settings

Tables 6 and 7 show the input data and parameters of the testbed. Table 6 shows the parameters regarding the structure of CWs: (i) number of tasks, (ii) amount of instructions per task, (iii) required service to execute tasks, (iv) number of workflow segments, and (v) connectivity level. Table 7 shows the parameters regarding the agent-based testbed: (i) number of agents, (ii) workflow request rate, (iii) number of service types, (iv) processing capacities of the Cloud services, and (v) number of agents involved per CNP enactment.

Table 6

<table>
<thead>
<tr>
<th>Input data</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of workflow tasks</td>
<td>100</td>
</tr>
<tr>
<td>No. of instructions per task</td>
<td>1000 million of instructions</td>
</tr>
<tr>
<td>Required service type</td>
<td>{s5}</td>
</tr>
<tr>
<td>Workflow segments</td>
<td>{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}</td>
</tr>
<tr>
<td>Connectivity level</td>
<td>{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}</td>
</tr>
</tbody>
</table>

Table 7

<table>
<thead>
<tr>
<th>Input data</th>
<th>Possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent types</td>
<td>CAs, BAs, SPAs, RAs</td>
</tr>
<tr>
<td>No. of agents</td>
<td>10, 10, 10, 1000</td>
</tr>
<tr>
<td>Workflow request rate</td>
<td>500 ms</td>
</tr>
<tr>
<td>No. of service types</td>
<td>1</td>
</tr>
<tr>
<td>(service type, processing capacity)</td>
<td>(s5, 1000)</td>
</tr>
<tr>
<td>Agents involved per CNP</td>
<td>4</td>
</tr>
</tbody>
</table>
CWs were defined to contain 100 tasks to emphasize any possible change that could be derived from their random creation. CWs ranging from 1 to 10 segments and from 1 to 10 connectivity levels were created. In doing so, the agent-based workflow execution approach was evaluated using random pipeline-like CWs (CWs with 10 segments and a connectivity level of 1) and parallel-like CWs (CWs with 1 segment and a connectivity level of 1). In addition, CWs of different connectivity levels were included from 1 ordering constraint to 10 ordering constraints for each number of workflow segments.

Tasks and services were homogeneous to avoid any possible noise caused by slow services executing big tasks (see Observation 3 of Section 5.1).

The remaining parameters have the same justification presented in Section 5.1.

For each configuration of the testbed, 100 experiment runs were carried out, one for each combination of number of segments and connectivity level. Each run consisted of 10 concurrent executions resulting in a total of 1000 Cloud workflow executions.

Performance measures

The performance measures are: (i) percentage of successful workflow executions, (ii) average workflow makespan, and (iii) average number of messages exchanged. See Table 5 for details.

Simulation results are shown in Figs 11 and 12. From these results, four observations were drawn.

Observation 1

Given randomly created CWs with different connectivity levels and parallelism levels, the agents achieved a 100% success rate in executing the CWs in a concurrent manner.

Analysis

The agents achieved a 100% success rate in executing CWs for all the (i) connectivity levels and (ii) parallelism levels.

By adopting concurrent and parallel CNP synchronized by CWs’ ordering constraints, all the agents in all the workflow executions were capable of contracting proper SPAs that executed the workflow tasks even when dealing with a broad variety of parallelism levels. In addition, as pointed out in Observation 1 of Section 5.1, the CAs could contract BAs to carry out the CW and the BAs could select different SPAs with appropriate resources to execute workflow tasks.

This result indicates that the agents achieved a 100% success rate in executing CWs because: (i) CAs had a high chance in finding BAs that could select and combine a group of SPAs with appropriate services to execute workflow tasks, and (ii) CAs, BAs, SPAs, and RAs can work effectively together to carry out the executions of randomly generated CWs.

Observation 2

In general, CWs with fewer segments, i.e., parallel-like workflows, were executed in shorter time than CWs with more segments, i.e., pipeline-like workflows. In addition, the makespan showed a linear-like pattern with respect to the number of workflow segments (parallelism level).

Analysis

As shown in Fig. 11, the CWs with a higher parallelism level (less segments) took shorter time to be executed than CWs with a lower parallelism level (more segments). This is because the parallelism level of CWs, i.e., the number of workflow tasks that can be executed in parallel depends on the number of seg-
ments. The more segments a CW has, the less task parallelization can be achieved because tasks have more precedence constraints, which serializes their executions. Hence, CWs with fewer segments and less precedence constraints allowed more tasks to be executed in parallel at a time than CWs with more segments.

The ascending linear-like pattern of makespan with respect to the number of segments was due to the length of paths from the initial node to the final node (e.g., an 8-segment CW had on average 8-task long paths from the initial node to the final node). Longer time was needed to execute CWs that had longer paths due to the serialization induced by more precedence constraints.

This result indicates that the agents were capable of parallelizing the execution of work flows in an efficient manner taking into consideration the restrictions established by the ordering constraints.

**Observation 3**

On the whole, CWs with less ordering constraints were executed in shorter times than CWs with more ordering constraints.

**Analysis**

Figure 12 shows that, in general, the average makespan increased with the number of ordering constraints. This is because workflow tasks with more ordering constraints had to wait for more tasks to be executed in order to enable their executions. In Fig. 12, the point corresponding to connectivity level 8 deviated from the overall tendency. This deviation was due to the random creation of CWs consisting of assigning up to $n$ ordering constraints to each task, e.g., a task with a connectivity level of 3 has 3 attempts to connect with 3 tasks. However, if a task to connect with was selected more than once, it counted as an attempt, and thus its connections were reduced, causing the deviations shown in Fig. 12.

This result shows that the agents could efficiently synchronize and coordinate the execution of CWs with different connectivity levels. In general, the time needed to execute CWs depended on the number of ordering constraints.

**Observation 4**

The number of messages that the agents exchanged remained unchanged regardless of the connectivity level and the number of workflow segments (parallelism level) of CWs.

**Analysis**

Given a wide range of ordering constraints and parallelism levels, the agents sent a constant number of messages to execute randomly created CWs. As stated in Observation 5 of Section 5.1, the number of messages exchanged has a direct relation with respect to the number of workflow tasks. In addition, BAs handled the parallelization and synchronization of CWs internally, i.e., BAs kept track of the workflow executions, and parallelized (when possible) the assignment of tasks to SPAs based on the ordering constraints of CWs. Furthermore, SPAs handled the parallel assignation of requirements to RAs as well as the synchronization of RAs.

The agents were capable of effectively parallelizing and synchronizing the execution of CWs by sending a constant number of messages and by autonomously reacting to the completion of workflow tasks.
6. Discussion

Since this work focuses on agent-based Cloud workflow execution, the areas to discuss are: (i) resource allocation, (ii) Cloud service composition, and (iii) workflow execution engines.

Resource allocation mechanisms adopting central allocators [3, 20] are commonly used in closed and controlled environments due to the fact that central allocators require complete control and knowledge about existing services and their capabilities. In addition, central entities usually become system bottlenecks. In distributed resource allocation mechanisms, no central allocator exists but a set of distributed allocators (agents) that interact among them to achieve resource allocations. These mechanisms range from auction [17, 18] to bargaining methods [29]. However, as the resource allocation methods become more complex, additional and more sophisticated agent capabilities are required. In this regard, the CNP [30] is probably the most commonly adopted resource allocation system, given its effectiveness.

Cloud service composition has been investigated in [10, 35, 36]. In [35, 36], complete knowledge of all services is assumed. Moreover, the composition process is centralized, and only atomic web services are considered. In contrast to [35] and [36], this work provides a decentralized composition for both atomic and complex services, i.e., services that may adopt an interaction protocol to fulfill their requirements. In [10], an agent-based Cloud service composition approach endowed with the CNP and acquaintance networks was presented. Although the agents in [10] can achieve service compositions with very few knowledge about the services and their capabilities, the number of messages exchanged is large due to the interaction required by the agents’ self-organizing skills. In addition, new issues such as creating and maintaining acquaintance networks are introduced.

Unlike workflow engines that require complete knowledge about providers, services and their current status (see [16]), the agent-based workflow execution approach in this work runs CWs based on the autonomous and cooperative interactions of agents without the need to fully planning ahead. In addition, the agents can handle \( n \) workflow executions concurrently in contrast to existent workflow engines: SwinDeWG [32], Pegasus [16], and GridBus [33] where only one workflow can be executed at a time. Moreover, in this present work, agent communication is direct, i.e., agents communicate directly among each other, removing possible bottlenecks and unnecessary messages, unlike GridBus [33] that centralizes communication using an event service server.

Finally, it is acknowledged that this present research is a significantly and considerably extended version of the work presented in [11]. Whereas the scope of [11] was limited to only Cloud service composition, in this work, a more complex problem is addressed: Cloud workflow execution. This work has augmented [11] as follows:

- CW object nets that provide constraints to concurrent and parallel service compositions to execute workflow tasks handled by BAs were defined (Section 4.5). Both BA and SPA system nets as well as both CW and RA object nets were combined to create a workflow execution engine (Section 4). In addition, both the agent models and the agent-based testbed support concurrent service selection, concurrent service composition, and concurrent execution of CWs. Furthermore, results obtained in [11] were augmented and generalized by conducting considerably more experiments in a larger scale (e.g., using 1030 agents) with much more stringent settings.

7. Conclusion and future work

The novelty and significance of this work is that, to the best of the authors’ knowledge, it is the earliest initiative that adopts an agent-based approach for supporting Cloud workflow execution in one or multiple Clouds, using (i) dynamic resource allocation, (ii) dynamic service composition, and (iii) coordination of distributed and self-interested participants.

The contributions of this paper are as follows:

1) Whereas [16, 32, 33] only execute one workflow at a time in only one Cloud provider, this work has developed an agent-based Cloud workflow execution approach (Sections 3 and 4) capable of dynamically allocating and composing a collection of Cloud resources from multiple Cloud providers to execute \( n \) CWs in a concurrent and parallel manner.

2) Formal Petri-net based methodologies (Section 4) to design Cloud resources (Section 4.4) and CWs (Section 4.5) that sustain concurrent and parallel management of workflow executions were defined.

3) A testbed (Section 3) to simulate distributed workflow execution where agents, equipped with the CNP as a cost-based service selection mech-
anism (absent in [16,32,33]), can autonomously and self-interestedly execute CWs was developed.

4) Simulation results (Section 5) showed that: (i) The agents were capable of successfully executing randomly created CWs in a parallel and concurrent manner. (ii) Cloud workflow execution was efficiently achieved since the makespan and number of messages exchanged only increased linearly with the number of tasks. (iii) Task parallelization was efficiently achieved in randomly created CWs with diverse levels of parallelism and ordering constraints.

Future research directions include: (i) Endowing the workflow engine with scheduling policies that allow agents to execute CWs in a given pool of Cloud resources. (ii) Enriching the service selection features, e.g., service reliability, so multiple-issue negotiation of quality of service parameters may take place. (iii) Developing a multi-objective optimization approach to determine the minimum set of Cloud resources to execute CWs constrained by consumers’ budget and deadlines. (iv) Endowing the agents with exception handling mechanisms to support Cloud workflow execution. (v) Deploying the agent-based Cloud workflow execution architecture in commercial Clouds and conducting exhaustive and large scale experiments to evaluate the scalability of the agent-based Cloud workflow execution method.

Acknowledgment

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MEST) (KRF-2009-220-D00092) and the DASAN International Faculty Fund (project code: 140316). The authors would like to thank the Editor-in-Chief and the referees for their comments and suggestions.

References


