Future Generation Computer Systems 29 (2013) 1682–1699

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A family of heuristics for agent-based elastic Cloud bag-of-tasks concurrent scheduling

J. Octavio Gutierrez-Garcia, Kwang Mong Sim *

[a] Department of Information and Communications, Gwangju Institute of Science and Technology, Gwangju 500-712, Republic of Korea

[b] School of Computing, University of Kent, Medway Building, Chatham Maritime, Kent ME4 4AG, United Kingdom

ARTICLE INFO

Article history:
Received 15 June 2011
Received in revised form
2 November 2011
Accepted 18 January 2012
Available online 7 February 2012

Keywords:
Scheduling heuristics
Bag-of-tasks applications
Cloud computing
Multi-agent systems
Agent-based Cloud computing
Elastic Cloud resource allocation

ABSTRACT

The scheduling and execution of bag-of-tasks applications (BoTs) in Clouds is performed on sets of virtualized Cloud resources that start being exhausted right after their allocation disregarding whether tasks are being executed. In addition, BoTs may be executed in potentially heterogeneous sets of Cloud resources, which may be either previously allocated for a different and fixed number of hours or dynamically reallocated as needed. In this paper, a family of 14 scheduling heuristics for concurrently executing BoTs in Cloud environments is proposed. The Cloud scheduling heuristics are adapted to the resource allocation settings (e.g., 1-hour time slots) of Clouds by focusing on maximizing Cloud resource utilization based on the remaining allocation times of Cloud resources. Cloud scheduling heuristics supported by information about BoT tasks (e.g., task size) and/or Cloud resource performances are proposed. Additionally, scheduling heuristics that require no information of either Cloud resources or tasks are also proposed. The Cloud scheduling heuristics support the dynamic inclusion of new Cloud resources while scheduling and executing a given BoT without rescheduling. Furthermore, an elastic Cloud resource allocation mechanism that autonomously and dynamically reallocates Cloud resources on demand to BoT executions is proposed. Moreover, an agent-based Cloud BoT scheduling approach that supports concurrent and parallel scheduling and execution of BoTs, and concurrent and parallel dynamic selection and composition of Cloud resources (by making use of the well-known contract net protocol) from multiple and distributed Cloud providers is designed and implemented. Empirical results show that BoTs can be (i) efficiently executed by attaining similar (in some cases shorter) makespans to commonly used benchmark heuristics (e.g., Max–min), (ii) effectively executed by achieving a 100% success execution rate even with high BoT execution request rates and executing BoTs in a concurrent and parallel manner, and that (iii) BoTs are economically executed by elastically reallocating Cloud resources on demand.

1. Introduction

Bag-of-tasks applications (BoTs) are sets of numerous unconnected (i.e., without precedence constraints) tasks, which can be highly parallelized given their unconnected nature. Scheduling and executing BoTs in Cloud environments is performed on sets of virtualized Cloud resources, which are allocated in terms of fixed and predefined allocation slots (e.g., 1-hour time slots in Amazon EC2 [1]) that start being exhausted right after their allocation disregarding whether tasks are being executed. In addition, BoTs may be executed in potentially heterogeneous sets of Cloud resources, which may be either previously allocated for a different and fixed number of hours or dynamically reallocated as needed (taking advantage of the elasticity of Cloud environments).

The scheduling of independent tasks in a set of heterogeneous computing resources has been shown to be an NP-complete problem [2]. For this reason, many heuristics have been proposed, from low level execution of tasks in multiple processors [3,4] to high level execution of tasks in Grid and Cloud environments [5–9]. Scheduling heuristics can be classified into: immediate and batch mode scheduling heuristics [10]. Immediate mode scheduling heuristics map BoT tasks to Cloud resources as soon as they arrive at the scheduler, e.g., the first-come–first-served scheduling heuristic. Batch mode scheduling heuristics pre-schedule BoT tasks on a previously defined set of Cloud resources before starting the execution, e.g., the Min–min and Max–min scheduling heuristics (first proposed in [4] and implemented in Grid-like settings in [11]). However, the majority of the scheduling heuristics (both immediate and batch modes) presumes that BoT holders are only charged for task execution time or not charged at all, when in

* Corresponding author.
E-mail addresses: joseogg@gmail.com (J.O. Gutierrez-Garcia), prof_sim_2002@yahoo.com (K.M. Sim).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.01.005

Cloud environments, consumers are charged for complete (1-hour) allocation slots.

In this paper, a family of 14 Cloud scheduling heuristics (including both immediate and batch mode scheduling heuristics) based on the remaining allocation times of Cloud resources is proposed. The scheduling heuristics consist of two phases: task ordering, where tasks are ordered prior to execution (when possible), and task mapping, where tasks are mapped to available (unoccupied) Cloud resources. The Cloud scheduling heuristics aim to maximize resource utilization. Scheduling heuristics supported by available information about BoT tasks (e.g., task size) and/or Cloud resource performances are proposed. However, scheduling heuristics that require no information of either Cloud resources or tasks are also proposed.

Also, in this paper, an elastic Cloud resource allocation mechanism that reallocates Cloud resources as needed by the execution of BoTs is proposed. The elastic resource allocation mechanism consists of monitoring the remaining allocation times of Cloud resources as well as their statuses (e.g., busy executing tasks), and determining whether Cloud resources should be reallocated. In addition, the elastic Cloud resource allocation mechanism can dynamically reallocate Cloud resources without requiring information about task completion times or Cloud resources' computing capacities.

Both the Cloud scheduling heuristics and the elastic Cloud resource allocation mechanism were integrated into an agent-based approach for scheduling and executing BoTs in multiple Cloud providers in a concurrent and parallel manner. Agents are endowed with distributed and cooperative problem solving techniques (including the well-known contract net protocol (CNP) [12]) that allow automated and dynamic selection and composition of Cloud resources from a pool of Cloud providers to execute BoTs. In addition, the agent-based problem solving techniques support distributed, concurrent, and parallel scheduling and execution of BoTs in heterogeneous sets of dynamically provisioned Cloud resources allocated in terms of 1-hour time slots.

The significance of this work is that, to the best of the authors' knowledge, it is the earliest work in adopting an agent-based Cloud BoT concurrent scheduling and execution approach endowed with a family of both immediate and batch mode scheduling heuristics adapted to the resource allocation settings (e.g., 1-hour time slots in Amazon EC2 [1]) of Clouds. In addition, it is the earliest work in adopting an elastic Cloud resource allocation mechanism that autonomously and dynamically reallocates Cloud resources to BoT executions. Moreover, the family of Cloud scheduling heuristics supports the dynamic inclusion of new Cloud resources while scheduling and executing BoTs. The contributions of this work are as follows:

(1) Designing and implementing an agent-based testbed for scheduling and execution of BoTs in Cloud environments in a concurrent and parallel manner (Section 2).
(2) Engineering and integrating an elastic Cloud resource allocation mechanism into the agent-based Cloud BoT scheduling approach (Section 3).
(3) Devising and implementing a family of 14 Cloud scheduling heuristics adapted to the resource allocation settings of Clouds (Section 4).
(4) Providing experimental evidence to demonstrate (i) the efficiency of the Cloud scheduling heuristics (Section 5.1), (ii) the effectiveness of the agent-based Cloud BoT concurrent scheduling and execution approach (Section 5.2), and (iii) the efficiency of the elastic Cloud resource allocation mechanism (Section 5.3).

This paper is structured as follows. Section 2 describes the agent-based Cloud architecture for BoT scheduling and execution. Section 3 presents the elastic Cloud resource allocation mechanism. Section 4 includes the definition of the family of 14 Cloud scheduling heuristics. Section 5 presents the evaluation and simulation results of both the family of Cloud scheduling heuristics and of the agent-based Cloud BoT concurrent scheduling and execution approach, as well as the evaluation and simulation results of the elastic Cloud resource allocation mechanism. Section 6 includes a comparison with related work. Finally, Section 7 presents concluding remarks and future research directions.

2. Agent-based Cloud architecture for BoT scheduling and execution

Agents representing consumer, brokers, Cloud providers, and Cloud resources interact among themselves to dynamically select and compose the best (cheapest) available Cloud resources from a pool of heterogeneous Cloud providers to execute and schedule BoTs in a distributed manner. The distributed scheduling and execution of BoTs is supported by an agent-based Cloud architecture (Section 2.1) as well as by a set of agent interaction protocols (Section 2.2) including the well-known CNP.

2.1. Agent-based Cloud architecture

The agent-based Cloud architecture for BoT scheduling (Fig. 1) is composed of: a service ontology, web services, resource agents, service provider agents, broker agents, consumer agents, and a Cloud directory.

(1) The service ontology defines both functional and nonfunctional Cloud resource capabilities. A functional capability defines the functions performed by Cloud resources, e.g., rendering service. A nonfunctional capability defines how Cloud resources execute their functions, e.g., computing capacity.
(2) Web services are interfaces that provide remote access to Cloud resources. Web services are described by the descriptions of the Cloud resources to which they provide access.
(3) Resource agents (RAs) wrap and orchestrate web services. RAs are enlisted in service provider agents.
(4) Service provider agents (SPAs) manage a set of RAs and offer for lease Cloud resources to broker agents. In addition, SPAs map service capabilities enlisted in the service ontology with their RAs' capabilities by performing a one-to-one matching. Moreover, SPAs handle dynamic and elastic Cloud resource reallocation to execute BoTs (see Section 3 for details) by interacting with broker agents.
(5) Broker agents (BAs) offer for lease BoT execution and scheduling services to consumer agents. BAs compose Cloud resources from multiple SPAs, and then schedule for execution consumer agents' BoTs in their corresponding previously composed sets of Cloud resources. In addition, BAs handle dynamic and elastic Cloud resource reallocation to execute BoTs (see Section 3 for details) by interacting with SPAs.
(6) Consumer agents (CAs) submit BoTs to BAs and map BoT tasks to available Cloud resource types by performing a one-to-one matching between tasks' requirements and available Cloud resource types.
(7) The Cloud directory is a listing of BAs' and SPAs' addresses as well as their functional and nonfunctional capabilities. BAs' capabilities are denoted as *broker*, and SPAs' capabilities are described by the functional and nonfunctional capabilities of their enrolled RAs. The Cloud directory is provided by a system agent, to which BAs and SPAs register, and CAs and BAs consult to look for BAs and SPAs, respectively.

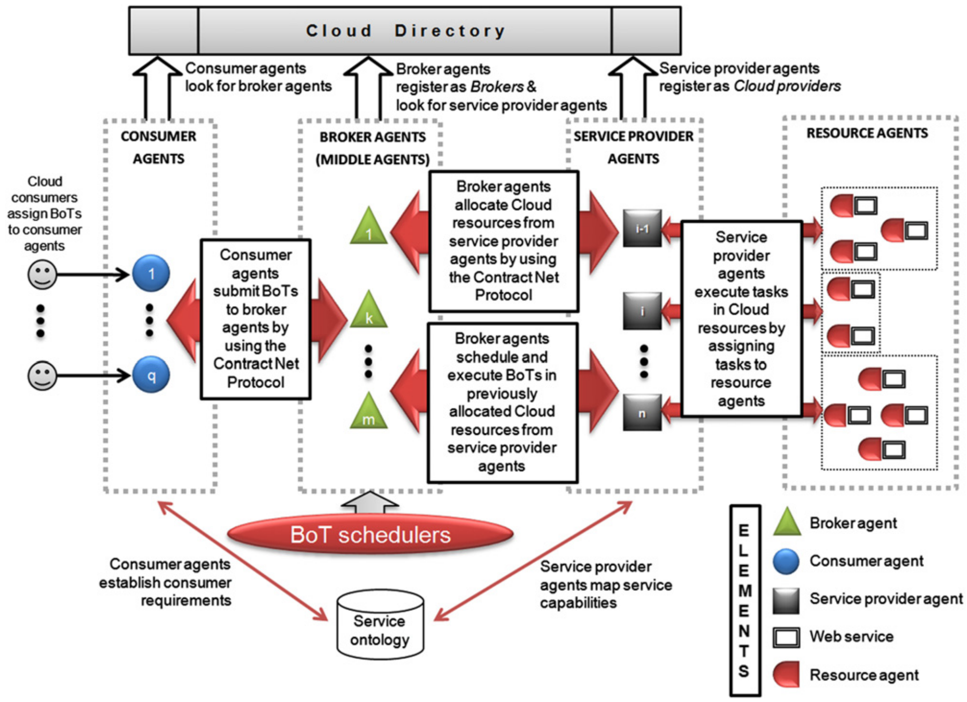

Cloud Directory
Consumer agents look for broker agents
Broker agents register as Brokers & look for service provider agents
Service provider agents register as Cloud providers
CONSUMER AGENTS
BROKER AGENTS (MIDDLE AGENTS)
SERVICE PROVIDER AGENTS
RESOURCE AGENTS
Cloud consumers assign BoTs to consumer agents
Consumer agents submit BoTs to broker agents by using the Contract Net Protocol
Broker agents allocate Cloud resources from service provider agents by using the Contract Net Protocol
Broker agents schedule and execute BoTs in previously allocated Cloud resources from service provider agents
Service provider agents execute tasks in Cloud resources by assigning tasks to resource agents
BoT schedulers
Consumer agents establish consumer requirements
Service ontology
Service provider agents map service capabilities
ELEMENTS
Broker agent
Consumer agent
Service provider agent
Web service
Resource agent

Fig. 1. Agent-based Cloud architecture for BoT scheduling.

2.2. *Agent-based Cloud BoT scheduling and execution*

Interaction among CAs, BAs, and SPAs is based on the CNP. The CNP has two roles: initiator (or manager) and participant (or contractor). An agent (e.g., a CA) playing the initiator role of the CNP sends a *call-for-proposals* message containing a task to be executed (e.g., a BoT) to n participant agents (e.g., BAs). Then, the participant agents reply with either a *refuse* message to discard the *call-for-proposals* or a *proposal* message containing a bid for executing the task (e.g., overall cost for executing a BoT) to participate in the bid process. From the m ($m < n$) participant agents that replied to the *call-for-proposals* message, the initiator agent selects the best (e.g., cheapest) proposal and sends an *accept-proposal* message to the winning participant agent, and $m - 1$ *reject-proposal* messages to the remaining participant agents. Subsequently, the winning participant agent executes the task and sends the output to the initiator agent.

A BoT scheduling scenario is as follows (see Fig. 2): firstly, a CA submits a BoT to BAs by adopting the initiator role of the CNP (the CA's *call-for-proposals* message include the BoT to be executed) with BAs as participants (BAs' proposals include the overall cost for executing the BoT). Then, the CA selects the cheapest BA to execute the BoT. Afterward, the winning BA composes a set of Cloud resources from multiple SPAs by adopting parallel CNPs (one for each Cloud resource to be allocated) with SPAs as participants. The BA's *call-for-proposals* message includes a Cloud resource type and the number of hours to be allocated, and the SPAs' proposals contain the allocation cost of a given Cloud resource type, for a given number of hours. The resultant set of winning SPAs allocate the Cloud resources via their RAs by sending allocation request messages. Once all the Cloud resources are allocated, the BA schedules (by using one of the 14 scheduling heuristics, see Section 4 for details) and executes the BoT in the previously composed set of Cloud resources by delegating the BoT tasks to SPAs, which in turn, delegate the tasks to their RAs. Then, the RAs execute the tasks and return the outputs to the SPAs. SPAs forward the BoT tasks' outputs to the BA, which receives and collects all the tasks' outputs for later delivery to the CA.

3. Individualized elastic Cloud resource allocation mechanism

The individualized elastic Cloud resource allocation mechanism (I-ElasticAM) is a distributed and cooperative problem solving technique to reallocate Cloud resources on demand for executing BoTs based on dynamic information about Cloud resources' statuses (e.g., idle or busy executing tasks).

There are two main agent behaviors (i.e., agent functionalities implemented as threads) in I-ElasticAM: I-ElasticAM_BA (see Fig. 3) and I-ElasticAM_SPA (see Fig. 4), which are built into BAs and SPAs, respectively.

(1) Behavior I-ElasticAM_BA (Fig. 3) is added to BAs for each Cloud resource initially allocated for 1 h to execute a given BoT. In doing so, an individualized elastic allocation mechanism is assigned to each Cloud resource, which may have different allocation times. Behavior I-ElasticAM_BA is implemented as a cyclic behavior, i.e., an agent behavior periodically activated by a time-driven event. The time-driven event activation (see the 5th input parameter in Fig. 3) of I-ElasticAM_BA is defined by the initial Cloud resource allocation time, the length of the allocation time slots (e.g., 1 h), and the reallocation threshold (i.e., amount of time required to reallocate Cloud resources). Thus, a behavior I-ElasticAM_BA is activated (by a time-driven event activation) just before the allocation time of its corresponding Cloud resource expires. Then, the BA verifies whether the Cloud resource is idle or busy executing a previously assigned task (see line 2 of Fig. 3). If the Cloud resource is idle, i.e., it is not currently executing any task, the behavior is terminated (see line 3 of Fig. 3), under the assumption that idle Cloud resources are no longer receiving tasks for execution and therefore no reallocation is needed anymore. On the other hand, if the Cloud resource is busy executing tasks (see line 4 of Fig. 3), the BA (i) requests the reallocation of the Cloud resource to the corresponding SPA, (ii) waits for an *acknowledgment* message, (iii) updates the corresponding contract, and (iv) establishes a new time-driven event activation for the behavior I-ElasticAM_BA (see lines 5–8 of Fig. 3).

It should be noted that behavior I-ElasticAM_BA was designed to be periodically activated based on the allocation time of

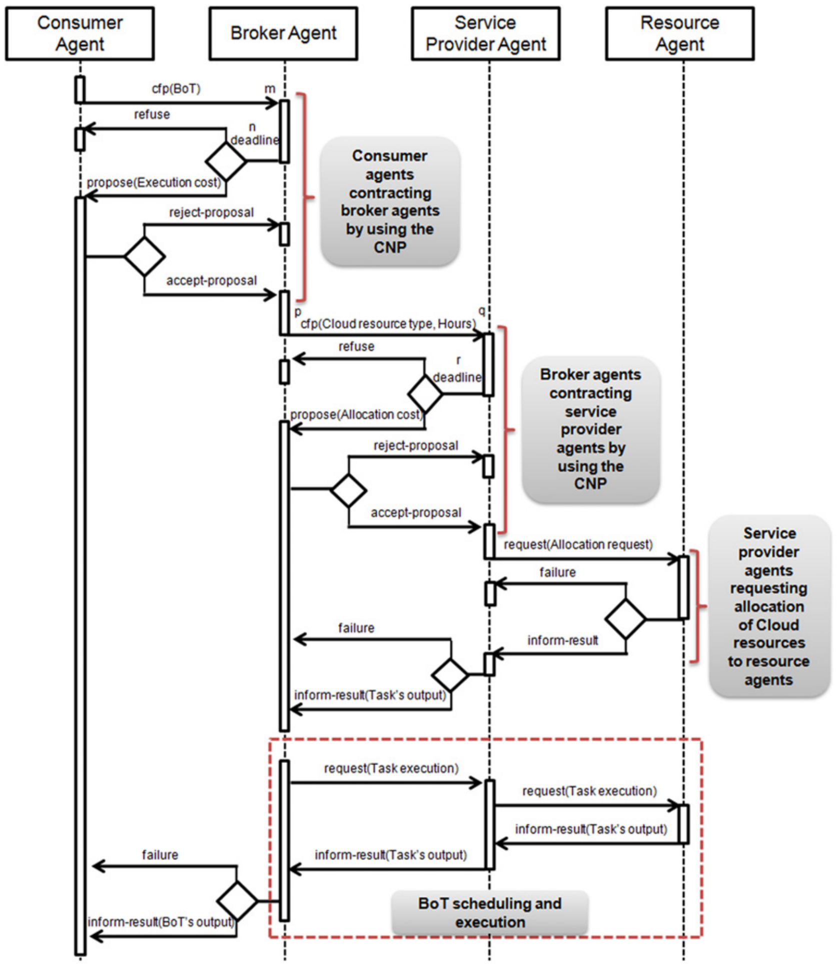

Consumer Agent
Broker Agent
Service Provider Agent
Resource Agent
cfp(BoT)
m
refuse
n
deadline
propose(Execution cost)
reject-proposal
accept-proposal
Consumer agents contracting broker agents by using the CNP
p
q
cfp(Cloud resource type, Hours)
refuse
r
deadline
propose(Allocation cost)
reject-proposal
accept-proposal
Broker agents contracting service provider agents by using the CNP
request(Allocation request)
failure
inform-result
Service provider agents requesting allocation of Cloud resources to resource agents
failure
inform-result(Task's output)
request(Task execution)
request(Task execution)
inform-result(Task's output)
inform-result(Task's output)
failure
inform-result(BoT's output)
BoT scheduling and execution

Fig. 2. Agent interaction protocols for Cloud BoT scheduling and execution.

```
Input:
  • A Cloud resource ID,
  • An SPA ID (from which the Cloud resource was allocated),
  • A contract ID that states the agreement between the BA and the corresponding SPA,
  • A reallocation threshold,
  • A time-driven event activation (behavior's cycle duration periodic time) =
    Initial Cloud resource allocation time + length of allocation time slots - reallocation threshold

Algorithm:
1:  Get current status of the Cloud resource via its ID
2:  if (the Cloud resource is idle) then
3:      Terminate behavior I-ElasticAM_BA
4:  else /* if the Cloud resource is busy executing a previously assigned BoT task */
5:      Request reallocation of the Cloud resource to the SPA
6:      BlockReceive(Acknowledgment message) from the SPA
7:      Update the corresponding contract established between the BA and the SPA
8:      Establish a new time-driven event activation for the behavior I-ElasticAM_BA =
        new starting Cloud resource allocation time + length of allocation time slots –
        reallocation threshold

Output:
  • Request for reallocation of a given Cloud resource for the next time slot, or
  • Behavior termination
```

Fig. 3. Agent behavior I-ElasticAM_BA.

Cloud resources, so BAs can execute BoTs (supported by elastic resource allocation) with Cloud scheduling heuristics that are either endowed with or not endowed with information about task completion times and/or Cloud resources' computing capacities, as empirically demonstrated in Section 5.3.

(2) Behavior I-ElasticAM_SPA (Fig. 4) is added to each SPA to handle reallocation requests from BAs, regarding currently allocated Cloud resources. Behavior I-ElasticAM_SPA is activated by the reception of reallocation requests from BAs (see line 1 of Fig. 4). Then, the SPA (i) updates the allocation record of the corresponding Cloud resource, (ii) charges the BA based on the hourly cost rate assigned to the Cloud resource, (iii) updates

```
Input:
  • A request message from a BA asking for reallocation of a previously
    contracted Cloud resource for the next time slot

Algorithm:
1:  if BlockReceive(request for reallocation) then
2:      Update allocation record of the given Cloud resource
3:      Charge the BA based on the hourly cost rate assigned to the Cloud resource
4:      Update the corresponding contract established between the SPA and the BA
5:      Send acknowledgement message

Output:
  • A reallocation of the Cloud resource for the next time slot
```

Fig. 4. Agent behavior I-ElasticAM_SPA.

the corresponding contract, and (iv) sends an *acknowledgment* message to the BA (see lines 2–5 of Fig. 4).

4. Cloud scheduling heuristics

The Cloud scheduling heuristics are focused on maximizing Cloud resource utilization based on the remaining allocation times of Cloud resources by taking advantage of the Cloud resource allocation settings (e.g., 1-hour time allocation slots). The Cloud scheduling heuristics are supported by information about BoT tasks (e.g., task size) and/or Cloud resource performances.

The Cloud scheduling heuristics consist of two phases: task ordering (Table 1) and task mapping (Table 2). The ordering types of BoT tasks are:

(1) *Unordered* (U). Tasks are sent for execution in an unordered manner whenever no information about the tasks is available

Table 1
Task ordering types.

ID	Task ordering type
U	Unordered tasks
LtoS	Ordered by size from large to small
StoL	Ordered by size from small to large

Table 2
Task mapping policies.

ID	Mapping policy
R	Random
MaxET	Maximum expected remaining allocation time
MaxCT	Maximum current remaining allocation time
MinET	Minimum expected remaining allocation time
MinCT	Minimum current remaining allocation time

either because not all the tasks are available at the moment of starting the execution or no information is disclosed by BoT holders. By executing tasks in an unordered manner, BoTs can be executed without information about expected task execution times. Cloud scheduling heuristics using the U ordering type are immediate mode scheduling heuristics.

(2) *Tasks ordered by size from large to small* (LtoS). When information about tasks (e.g., number of instructions, which can be used to compute the expected execution time of tasks in a heterogeneous pool of Cloud resources) is available, tasks are sorted from large to small tasks, with larger tasks being executed first. By executing larger tasks first, a higher parallelization of execution of tasks may take place, since many small tasks can be parallely executed at the same time that large tasks are being executed. Cloud scheduling heuristics using the LtoS ordering type are batch mode scheduling heuristics.

(3) *Tasks ordered by size from small to large* (StoL). Tasks are ordered from small to large, with smaller tasks being executed first. By executing smaller tasks first, a faster response rate from Cloud providers may be obtained. Cloud scheduling heuristics using the StoL ordering type are batch mode scheduling heuristics.

The task mapping policies are based on the remaining allocation time of Cloud resources. The task mapping policies in Cloud resources are:

(1) *Random mapping* (R). Tasks are executed in randomly selected and available (unoccupied) Cloud resources whenever no information about Cloud resource performances is available or Cloud resources have high variations on performance. By executing tasks in randomly selected Cloud resources, BoTs can be executed without information about Cloud resource performances.

(2) *Maximum expected remaining allocation time mapping* (MaxET). Tasks are executed in Cloud resources with the maximum expected remaining allocation time after assigning a given task. The expected completion time of tasks in a given Cloud resource is taken into account whenever information of both tasks and Cloud resource performances is available. The MaxET mapping policy was designed to make use of Cloud resources that have been allocated for more hours under the assumption that such resources are capable of executing more and longer tasks.

(3) *Maximum current remaining allocation time mapping* (MaxCT). Tasks are executed in Cloud resources with the maximum current remaining allocation time before assigning a given task, i.e., the expected completion time of tasks is not taken into account either because no task information or no Cloud resource performances are available. Similar to MaxET, MaxCT favors the use of Cloud resources that have been allocated for more hours. However, MaxCT does not require information about Cloud resource performances.

```
Input:
  • A BoT represented by a set of tasks $T$
  • A set of previously composed Cloud resources $CR$
  • A type of task ordering $\alpha \mid \alpha \in \{U, LtoS, StoL\}$
  • A type of mapping policy $\beta \mid \beta \in \{R, MaxET, MaxCT, MinET, MinCT\}$

Algorithm:
1:  BA sorts $T$ using $\alpha$
2:  For all unexecuted tasks $t_i$ in $T$ do
3:      BA maps task $t_i$ onto the best unoccupied Cloud resource in $CR$ based on $\beta$
4:      BA sends task $t_i$ to the corresponding SPA for execution
5:      if (all the Cloud resources are occupied) then
6:          BA waits for any available Cloud resource

Output:
  • An executed BoT ($T$) following a ($\alpha$, $\beta$) heuristic
```

Fig. 5. General heuristic algorithm for agent-based Cloud scheduling of BoTs.

(4) *Minimum expected remaining allocation time mapping* (MinET). Tasks are executed in Cloud resources with the minimum expected remaining allocation time after assigning a given task, i.e., the expected completion time of tasks in a given Cloud resource is taken into account whenever information of both tasks and Cloud resource performances is available. MinET aims to maximize the utilization of Cloud resources that are about to expire.

(5) *Minimum current remaining allocation time mapping* (MinCT). Tasks are executed in Cloud resources with the minimum current remaining allocation time before assigning a given task, i.e., the expected completion time of tasks is not taken into account either because no task information or no Cloud resource performances are available. Similar to MinET, MinCT aims to maximize the utilization of Cloud resources that are about to expire. However, MinCT does not require information about Cloud resource performances because tasks are allocated to Cloud resources with the minimum current remaining allocation time regardless of the task completion times.

From the combination of the task ordering types (Table 1) and task mapping policies (Table 2), a total of 14 scheduling heuristics are obtained: $\{U, LtoS, StoL\} \times \{R, MaxET, MaxCT, MinET, MinCT\}/(U, R)$. The (U, R) heuristic represents a completely random immediate mode scheduling heuristic that follows a first-come–first-served allocation policy. Therefore, the (U, R) heuristic is used for comparison and detailed evaluation of the agent-based scheduling (see Section 5.1, incise b).

BAs make use of the Cloud scheduling heuristics to execute BoTs. The general algorithm used by BAs for the two-phase Cloud scheduling heuristics is shown in Fig. 5. The Cloud scheduling algorithm is as follows. The scheduling algorithm receives as input four parameters: a BoT, a set of previously composed Cloud resources where the BoT is going to be executed, a type of task ordering (either U, LtoS or StoL), and a type of mapping policy (either R, MaxET, MaxCT, MinET or MinCT). Then, a BA sorts a set of BoT tasks based on the selected task ordering type (line 1 in Fig. 5) and for all the unexecuted BoT tasks (line 2 in Fig. 5), the BA maps each task onto the best unoccupied Cloud resource based on the selected task mapping policy (line 3 in Fig. 5). Afterward, the BA sends for execution the recently mapped tasks to the corresponding SPAs (line 4 in Fig. 5), i.e., SPAs where the selected Cloud resources are located. Subsequently, when all the Cloud resources are busy executing tasks (line 5 in Fig. 5), the BA waits for the reception of tasks' outputs indicating that a Cloud resource is available (line 6 in Fig. 5). Right after, the BA proceeds to map the next unexecuted tasks onto the recently available Cloud resources and so on until all the BoT tasks are completely executed giving as a result a completely executed BoT.

Since the 14 Cloud scheduling heuristics are based on selecting either the largest or smallest tasks or the maximum or minimum remaining allocation time of Cloud resources, the time complexity of the scheduling heuristics is mainly based on the complexity of a sorting algorithm to select either the first or last element of an array. For programming the 14 Cloud scheduling heuristics, the *mergesort* algorithm was used, which has a time complexity of $O(x * \log(x))$ [13], where x stands for number of array elements. Therefore, the time complexity of U-based ({U} × {MaxET, MaxCT, MinET, MinCT}) Cloud scheduling heuristics is bounded by $O(m * n * \log(n))$, where m stands for number of tasks contained in a given BoT and n stands for number of Cloud resources where a given task can be assigned. The second factor $(n * \log(n))$ is included because only the set of Cloud resources is sorted, and the first factor (m) because for every task to be mapped, the current best (e.g., based on MaxET) Cloud resource is selected. The time complexity of R-based ({LtoS, StoL} × {R}) Cloud scheduling heuristics is bounded by $O(m * \log(m))$, because only BoT tasks are sorted, and the Cloud resources are arbitrarily selected among the unoccupied Cloud resources. Finally, the time complexity of the remaining Cloud scheduling heuristics ({LtoS, StoL} × {MaxET, MaxCT, MinET, MinCT}) is $O(m*\log(m) + m * n * \log(n))$ because BoT tasks are sorted only once, and the current best Cloud resource is selected for assigning each task.

5. Evaluation and results

Three series of experiments were carried out for evaluating: (i) the Cloud scheduling heuristics (Section 5.1), (ii) the agent-based concurrent BoT scheduling and execution (Section 5.2), and (iii) the agent-based Cloud BoT concurrent scheduling approach endowed with I-ElasticAM (Section 5.3), respectively. The experiments were conducted using the agent-based testbed (implemented using the java agent development framework (JADE) [14]) defined in Section 2.

The experiments were carried out on a computer with the following specifications: Intel Core 2 Duo E8500 3.16 GHz, 4 GB RAM, with a Windows Vista Enterprise (32 bits) operating system, service pack 2.

5.1. Evaluating the Cloud scheduling heuristics

(a) *Objective*. A series of experiments was designed to evaluate the efficiency of the 14 Cloud scheduling heuristics.

(b) *Benchmark scheduling heuristics*. A total of five commonly used scheduling heuristics [15,8,16,17] were used for benchmarking the proposed scheduling heuristics:

- *First-come–first-served* (FCFS) consisting of assigning BoT tasks (in an unordered manner) as soon as they are ready for execution to any available Cloud resource.
- *Greedy (response) scheduling* (Greedy-R) consisting of assigning BoT tasks with the quickest execution time first to the most powerful available Cloud resource to maximize system response time.
- *Greedy (parallelization) scheduling* (Greedy-P) consisting of assigning BoT tasks with the quickest execution time first to the less powerful available Cloud resource to maximize task parallelization as well as system response time.
- *Min–min heuristic* (Min–min) consisting of assigning BoT tasks to Cloud resources with the minimum earliest completion time first.
- *Max–min heuristic* (Max–min) consisting of assigning BoT tasks to Cloud resources with the maximum earliest completion time first.

The FCFS is an immediate mode scheduling heuristic that requires no information about tasks or Cloud resources. Conversely, Greedy-R, Greedy-P, Min–min, and Max–min are batch mode scheduling heuristics that require information about tasks and Cloud resources' computing capacities.

Section 5.1 includes a quantitative comparison between the benchmark scheduling heuristics and the proposed heuristics, see Section 6 for a qualitative comparison.

(c) *Experimental settings*. The testbed has three different sets of input parameters: (1) Cloud environment's input data (Table 3) consisting of nonfunctional types of Cloud resources (e.g., cluster) described by their computing capacities and hourly cost rates. (2) BoT-related input data (Table 4) consisting of number of tasks per BoT, number of instructions per task, probability distributions used for creating randomly generated BoTs, and a functional Cloud resource type, i.e., type of requirement that Cloud resources fulfill, e.g., encrypting/decrypting data. (3) Agent-based testbed parameters (Table 5) consisting of number of agents involved in the simulation grouped by agent type, number of agents involved per CNP execution, Cloud scheduling heuristics, a set of Cloud resources to be composed by BAs where BoTs were executed, and a simulation time rate.

Nonfunctional Cloud resource types (Table 3) were based on Amazon EC2 instance types [1] to have a heterogeneous pool of Cloud resources that can highlight the differences among the scheduling heuristics. In addition, only 7 out of 11 currently available Amazon EC2 instance types were considered because the remaining instances were considerably much more powerful than the selected instances types, e.g., clusters, and thus some of the scheduling heuristics may focus on such Cloud resources flattening the performance differences among them.

The number of tasks per BoT (Table 4) was set from 100 to 300 in 100-task increments to evaluate the performance with respect to the BoT size. The number of instructions per task was randomly selected from 400 to 4000 million of instructions in 400 million increments to create heterogeneous BoTs and provide a wide variety of scenarios to the scheduling heuristics. In addition, BoTs were randomly generated following four different probability distributions. Nonetheless, for the sake of fairness, the same randomly generated BoTs (of a given probability distribution and BoT size) were assigned to CAs, e.g., whenever agents executed a BoT of 200 tasks generated using a normal distribution, the BoT was the same.

BoTs were composed of tasks that required the same functional Cloud resource type to be executed: s_1, e.g., an encrypting service. In doing so, the evaluation was focused on scheduling rather than focusing on the composition and selection of heterogeneous functional Cloud resources.

The number of agents involved in the experiments was fixed to 5 CAs, 5 BAs, and 5 SPAs to focus the evaluation on scheduling rather than on interlayer agent interaction. RAs were fixed to 2500 to simulate the unlimited resources of Clouds. In addition, the nonfunctional Cloud resource types of RAs were randomly assigned from the available types (see Table 3) and their functional Cloud resource type was set to s_1. Moreover, RAs were randomly assigned to SPAs to create a heterogeneous and asymmetrical Cloud environment.

The number of agents involved per CNP was limited to 1 manager (this role is played by either CAs or BAs) and 3 contractors (these roles are played by either BAs or SPAs) following the price studies presented in [18] to have a sufficient price sample and avoid unnecessary message exchanges. In addition, the content of agents' proposals for executing either BoTs (in the case of BAs) or allocating a given Cloud resource (in the case of SPAs) was the same in all cases, and thus agents arbitrarily selected any of the agents that replied to a given call for proposals. In doing so, the evaluation was focused on scheduling rather than on service selection.

BAs allocated a highly heterogeneous set of Cloud resources for 10 h to execute the BoTs. In doing so, BAs had enough time to execute the BoTs, and the characteristics of the Cloud scheduling heuristics were highlighted due to the variety of computing capacities of the Cloud resources.

Table 3
Cloud environment's input data for Experiments 5.1–5.3.

Nonfunctional Cloud resource types	Computing capacity[a]	Hourly cost rate[b]
Standard–small	1.00	\$0.085
Standard–large	4.00	\$0.34
Standard–extra large	8.00	\$0.68
Micro	2.00	\$0.02
High-memory–extra large	6.50	\$0.50
High-memory–double extra large	13.00	\$1.00
High-CPU–medium	5.00	\$0.17

[a] Computing capacities were expressed in EC2 compute units (ECU) [1], which for experimental purposes were defined as 1 EC2 compute unit = 1,000,000 million of instructions per second.
[b] Hourly cost rates were expressed in USD and were based on the EC2 pricing model [1].

Table 4
BoT-related input data source for Experiment 5.1.

Input data	Possible values			
BoT size (No. of tasks per BoT)	{100, 200, 300} tasks			
Task size (No. of instructions per task)	{400, 800, 1200, 1600, 2000, 2400, 2800, 3200, 3600, 4000} million of instructions			
Probability distributions used for creating randomly generated BoTs	*Uniform distribution*. An equitable number of tasks of each task size category were contained in BoTs	*Normal distribution*. Many middle size tasks, and fewer big and small tasks were contained in BoTs	*Left half of a normal distribution*. Many big tasks and fewer small tasks were contained in BoTs	*Right half of a normal distribution*. Many small tasks and fewer big tasks were contained in BoTs
Functional Cloud resource type	Homogeneous: $\{s_1\}$			

Table 5
Agent-based testbed parameters for Experiment 5.1.

Input data	Possible values			
Agent types	CAs	BAs	SPAs	RAs
No. of agents	5	5	5	2500
Agents involved per CNP	1 manager and 3 contractors			
Cloud scheduling heuristics	{(U, MaxET), (U, MaxCT), (U, MinET), (U, MinCT), (LtoS, R) (LtoS, MaxET), (LtoS, MaxCT), (LtoS, MinET), (LtoS, MinCT), (StoL, R), (StoL, MaxET), (StoL, MaxCT), (StoL, MinET), (StoL, MinCT)} ∪ {FCFS, Greedy-R, Greedy-P, Min–min, Max–min}			
Cloud resources to be composed by BAs	{1 standard–small, 1 standard–large, 1 standard–extra large, 1 micro, 1 high-memory–extra large, 1 high-memory–double extra large, 1 high-CPU–medium} all of them allocated for 10 h			
Simulation time rate	60 s in simulation is equal to 1 h			

Table 6
Performance measure for Experiment 5.1.

Performance measure	Description
Av. BoT makespan (grouped by BoT size category and scheduling heuristic)	$\Sigma\ (M_{[\text{BoTsize},\text{Heuristic}]})/N_{[\text{BoTsize},\text{Heuristic}]}$
Performance measure's variables	**Description**
$M_{[\text{BoTsize},\text{Heuristic}]}$	Makespan of a BoT of size *BoTsize* executed using a given scheduling *Heuristic*
$N_{[\text{BoTsize},\text{Heuristic}]}$	No. of BoTs of size *BoTsize* executed using a scheduling *Heuristic*
Performance measure's constants	**Possible values**
BoT size category	{100, 200, 300} tasks
Heuristics	{(U, MaxET), (U, MaxCT), (U, MinET), (U, MinCT), (LtoS, R), (LtoS, MaxET), (LtoS, MaxCT), (LtoS, MinET), (LtoS, MinCT), (StoL, R), (StoL, MaxET), (StoL, MaxCT), (StoL, MinET), (StoL, MinCT), FCFS, Greedy-R, Greedy-P, Min–min, Max–min}

CAs submitted BoTs to BAs such that the BoTs were not executed simultaneously to avoid any possible noise (e.g., additional time) resulted from concurrent scheduling of BoTs (see Section 5.2 for an evaluation of the agent-based concurrent handling of BoT scheduling and execution). The simulation time rate was set to 1 h for every 60 s of simulation time for its convenience in conducting the experiments.

The 14 Cloud scheduling heuristics (proposed in Section 4) together with the benchmark scheduling heuristics (FCFS, Greedy-R, Greedy-P, Min–min, and Max–min) were used for executing the randomly generated BoTs (Table 4) using the randomly generated Cloud environments (Tables 3 and 5) for a total of 1140 scheduled and executed BoTs.

(d) *Performance measure*. The performance measure is average BoT makespan (amount of time required to execute a BoT). See Table 6 for details.

(e) *Results*. Experimental results are shown in Figs. 6–9. From these results, five observations were drawn.

Observation 1. In general, BoTs containing many small tasks and few big tasks scheduled using the MaxET task mapping policy ({U, LtoS, StoL} × {MaxET}) had the shortest makespans regardless of the task ordering type.

Analysis. As shown in Fig. 6 in the *overall* series, BAs scheduling right-half-normally distributed BoTs using the MaxET task mapping policy attained the shortest makespans. This is because the MaxET mapping policy was designed to execute

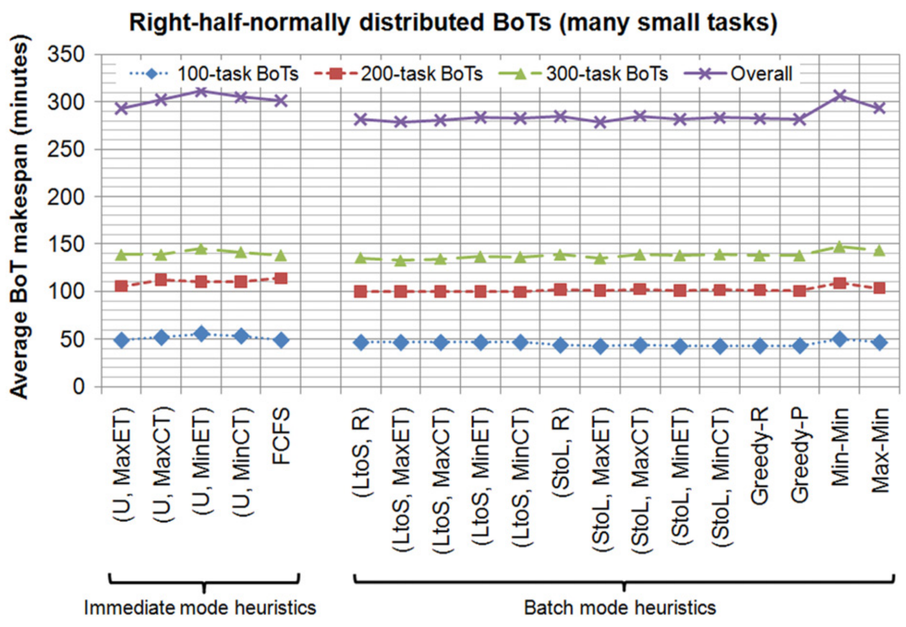


Fig. 6. Average makespan of right-half-normally distributed BoTs for all the heuristics.

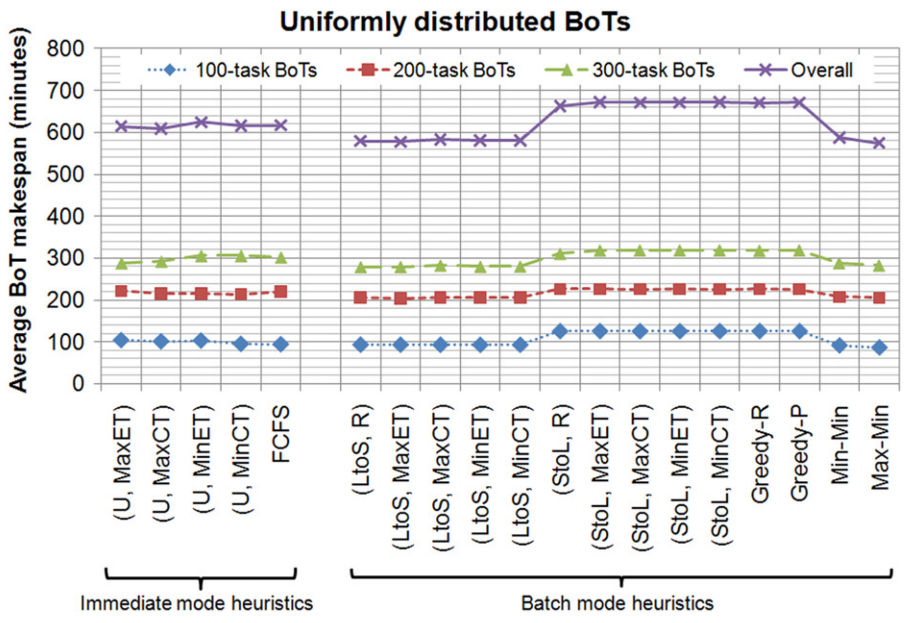


Fig. 7. Average makespan of uniformly distributed BoTs for all the heuristics.

tasks in Cloud resources with the maximum expected remaining allocation time after completing tasks, and since the Cloud resources were all allocated by the same number of hours, the maximum expected remaining allocation time implied that tasks were executed in the most powerful Cloud resources that provided the earliest task execution times. In addition, given that most of the tasks were small, the load unbalance produced by the few big tasks was quickly balanced by the remaining unexecuted small tasks.

MaxET-based scheduling heuristics not only had the shortest makespan in some scenarios (e.g., when scheduling right-half-normally distributed BoTs) by naturally balancing the load among Cloud resources but also they can be used with and without information about tasks and Cloud resource computing capacities in both immediate and batch scheduling modes and yet outperforming Greedy-R, Greedy-P, Min–min, and Max–min.

Observation 2. In general, BoTs containing uniformly distributed heterogeneous tasks (regarding size) scheduled using the LtoS task ordering type ({LtoS} × {R, MaxET, MaxCT, MinET, MinCT}) had the shortest makespans comparable to Min–min and Max–min regardless of the task mapping policy.

Analysis. As can be seen in Fig. 7 in the *overall* series, BAs scheduling uniformly distributed BoTs using the LtoS task ordering type attained among the shortest makespans. This is because the LtoS task ordering type was designed to schedule larger tasks first. In doing so, the big tasks were assigned to Cloud resources at the beginning of the BoT executions, and when the remaining unexecuted tasks in decreasing size were assigned, the load of Cloud resources was progressively balanced, achieving a performance better than Min–min and similar to Max–min (which was the best).

Scheduling uniformly (and right-half-normally) distributed BoTs using some of the heuristics supported by the remaining allocation time of Cloud resources can attain better or at least similar performances than commonly used benchmark scheduling

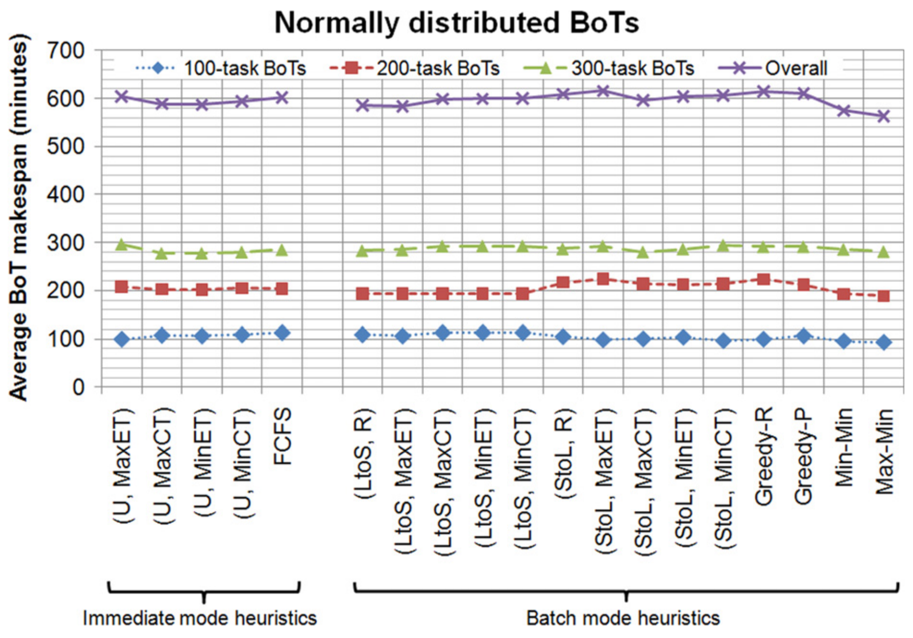


Fig. 8. Average makespan of normally distributed BoTs for all the heuristics.

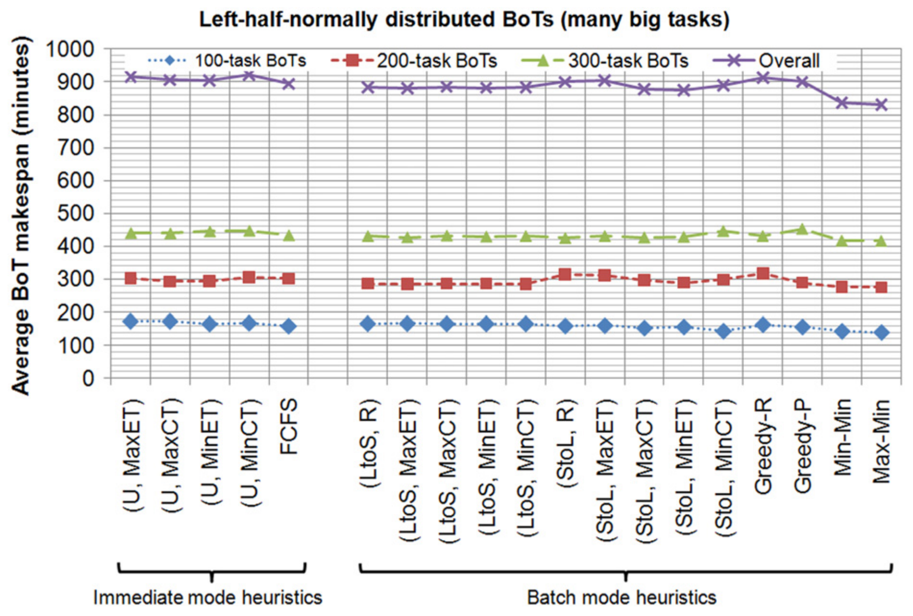


Fig. 9. Average makespan of left-half-normally distributed BoTs for all the heuristics.

heuristics: FCFS, Greedy-R, Greedy-P, Min–min, and Max–min, as suggested by Observations 1 and 2 of Experiment 5.1, and as shown in Figs. 6 and 7.

Observation 3. In general, the immediate mode scheduling heuristics ({U} × {MaxET, MaxCT, MinET, MinCT}) supported by the remaining allocation time of Cloud resources attained similar performances (in terms of makespan) among them and with respect to the FCFS' performance, regardless of the type of BoT scheduled (either uniformly, normally, right-half-normally, or left-half-normally distributed). In addition, the performance achieved by the immediate mode scheduling heuristics was similar (e.g., no more than a 10% difference in makespan) to the performance achieved by the batch mode scheduling heuristics.

Analysis. As shown in Figs. 6–9, the makespan attained by all the immediate mode scheduling heuristics was similar (except for some immediate mode scheduling heuristics when scheduling right-half-normally distributed BoTs, e.g., (U, MaxET) in Fig. 6, which considerably outperformed the other immediate mode scheduling heuristics). This is because the immediate mode scheduling heuristics' task mapping policies (MaxET, MaxCT, MinET, and MinCT) only take into consideration available Cloud resources for assigning a given task, i.e., tasks are mapped to Cloud resources that are available at the moment that the tasks are about to be sent for execution. However, usually only one Cloud resource was available at a time, given that Cloud resources had heterogeneous computing capacities and tasks with heterogeneous sizes were executed in an unordered manner. This caused completely different task completion times, and thus completely different availability times of the Cloud resources. Then, after the first round of task allocations (when all the Cloud resources were available), all the immediate mode scheduling heuristics scheduled tasks similarly, including FCFS.

As shown in Figs. 6–9, the performance of the ({U} × {MaxET, MaxCT, MinET, MinCT}) scheduling heuristics was close to the performance achieved by the batch mode scheduling heuristics, for instance, the overall makespan (i.e., the sum of all the BoTs' makespans) achieved by (U, MaxET) is only 6.33% higher than the overall makespan achieved by Max–min.

Despite the fact that in the immediate mode scheduling heuristics, only in the first round of task allocations, the task mapping policies are fully utilized, scheduling BoTs using the immediate mode scheduling heuristics can attain makespans close (only 6.33% higher) to those achieved by the batch mode scheduling heuristics. This is because the Cloud resources were continuously executing tasks, given that remaining unexecuted BoT tasks were promptly assigned (by the agents) to the Cloud resources as soon as the outputs of previously assigned tasks were received.

Observation 4. Due to the NP-complete nature of the scheduling problem [2], there was not dominant scheduling heuristics for all the BoT types (uniformly, normally, right-half-normally, and left-half-normally distributed BoTs), neither the proposed Cloud scheduling heuristics based on the remaining allocation time of Cloud resources nor the benchmark scheduling heuristics.

Analysis. As shown in Figs. 6–9, no heuristic dominated the others when scheduling randomly generated BoTs based on different probability distributions. This is because randomly generated BoTs using different probability distributions for determining the size of their tasks may generate advantageous scenarios for some heuristics (see the (StoL, MaxET) and Min–min heuristics in Figs. 6 and 7, respectively) and disadvantageous scenarios for other heuristics (see the Min–min and (StoL, MaxET) heuristics in Figs. 6 and 7, respectively). In addition, in some cases, the performance of the heuristics varied with respect to the other heuristics when executing BoTs of different sizes (see Fig. 6, 100-task and 200-task BoTs series for the StoL-based heuristics). This is because the performance of a given scheduling heuristic also depended on the particular characteristics of the randomly generated BoTs.

For certain cases, e.g., when scheduling right-half-normally distributed BoTs containing many small tasks and few big tasks (see Fig. 6), some of the proposed scheduling heuristics outperformed the benchmark scheduling heuristics. However, for certain cases (see Fig. 9) some benchmark scheduling heuristics, namely, Max–min and Min–min, scheduled left-half-normally distributed BoTs containing many big tasks and few small tasks outperforming the proposed scheduling heuristics. This is because the proposed scheduling heuristics could not balance the high load unbalanced caused by the big tasks (for left-half-normally distributed BoTs) with the task mapping policies (R, MaxET, MaxCT, MinET, MinCT) that decide what Cloud resource is selected to assign a task based on the available Cloud resources at a given moment (see line 6 of Fig. 5). In contrast, Max–min and Min–min computed the completion time for all the BoT tasks in all the Cloud resources and selected the tasks with the maximum and minimum completion time, respectively, before starting the BoT execution, preventing work load unbalances and attaining shorter makespans than the proposed scheduling heuristics for some cases (see Fig. 9).

Then, the selection of the most appropriate heuristic highly depends on the information available of tasks, Cloud resources, and the structure of BoTs. Nonetheless, the support provided by the remaining allocation time of Cloud resources provides additional qualitative advantages, e.g., dynamic inclusion of Cloud resources during BoT execution without rescheduling.

Observation 5. In general, the overall makespan attained for normally, right-half-normally, and left-half-normally distributed BoTs using batch mode scheduling heuristics supported by the remaining allocation time of Cloud resources ({LtoS, StoL} × {MaxET, MaxCT, MinET, MinCT}) did not have high variations (e.g., variations higher than 10% of the average makespan for each BoT type).

Analysis. In general, as shown in Figs. 6, 8 and 9, the makespans attained for normally, right-half-normally, and left-half-normally distributed BoTs scheduled using LtoS-based and StoL-based scheduling heuristics (i.e., batch mode scheduling heuristics) were similar. This is because the Cloud resources allocated for executing the BoTs had no idle time, i.e., the Cloud resources were executing BoT tasks at every moment (except for the almost negligible overhead time derived from agents preparing and scheduling BoT tasks, see Section 5.2, Observation 3), since the BAs (BoT schedulers) sent the tasks for execution as soon as the Cloud resources were available (see the general heuristic algorithm for agent-based Cloud scheduling in Fig. 5). In addition, since the Cloud resources had heterogeneous computing capacities and the tasks executed had heterogeneous sizes, very different task completion times were obtained, causing completely different availability times of the Cloud resources. Hence, usually only one Cloud resource was available at a time for executing a task, causing that the task mapping policies (R, MaxET, MaxCT, MinET, MinCT) behaved similarly, and thus similar makespans were obtained. Therefore, the small makespan differences across the batch mode scheduling heuristics (LtoS-based and StoL-based scheduling heuristics) were caused by the very last tasks executed in the Cloud resources.

It should be noted that, as shown in Fig. 7, the makespans attained for uniformly distributed BoTs scheduled using LtoS-based scheduling heuristics were not similar to the makespans attained using StoL-based scheduling heuristics. In fact, LtoS-based scheduling heuristics attained better performances (in terms of makespan) than StoL-based scheduling heuristics (see Fig. 7). This is because the BAs that scheduled the uniformly distributed BoTs using StoL-based scheduling heuristics sent for execution at the very end the biggest tasks, causing longer makespans when assigned to weak Cloud resources. In contrast, the BAs that scheduled the uniformly distributed BoTs using LtoS-based scheduling heuristics sent for execution at the very beginning the biggest tasks and then the remaining unexecuted tasks in decreasing size, progressively balancing the load of the Cloud resources, and thus attaining better performances, as explained in Observation 2 in Section 5.1.

The influence (i.e., the capacity to have an effect on BoT scheduling) that the task mapping policies (R, MaxET, MaxCT, MinET, MinCT) has for scheduling different types of BoTs (either uniformly, normally, right-half-normally, and left-half-normally distributed) is higher when there are several available Cloud resources to assign tasks (e.g., at the beginning of the BoT scheduling). However, as the number of available Cloud resources is reduced, the influence of a given task mapping policy in the BoT scheduling process is also reduced.

5.2. Evaluating the agent-based Cloud BoT concurrent scheduling

(a) *Objective*. A series of experiments was designed to evaluate the effectiveness of the agent-based Cloud BoT concurrent scheduling and execution approach.

(b) *Experimental settings*. The testbed has three different sets of input parameters: (1) Cloud environment's input data (Table 3), (2) BoT-related input data (Table 7), and (3) Agent-based testbed parameters (Table 8). Cloud environment's input data consists of nonfunctional types of Cloud resources described by their computing capacities and hourly cost rates. BoT-related input data consists of number of tasks per BoT, number of instructions per task, probability distribution used for creating randomly generated BoTs, and a functional Cloud resource type (e.g., a rendering service). Agent-based testbed parameters consist of number of agents involved in the simulation grouped by agent type, number of agents involved per CNP execution, a Cloud scheduling heuristic,

Table 7
BoT-related input data source for Experiment 5.2.

Input data	Possible values
BoT size (No. of tasks per BoT)	{50, 100, 150, 200, 250, 300} tasks
Task size (No. of instructions per task)	{400, 800, 1200, 1600, 2000, 2400, 2800, 3200, 3600, 4000} million of instructions
Probability distribution used for creating randomly generated BoTs	*Uniform distribution*. An equitable number of tasks of each task size category were contained in the BoTs
Functional Cloud resource type	Homogeneous: $\{s_1\}$

Table 8
Agent-based testbed parameters for Experiment 5.2.

Input data	Possible values			
Agent types	CAs	BAs	SPAs	RAs
No. of agents	{10, 20, 30}	5	5	2500
Agents involved per CNP	1 manager and 3 contractors			
Cloud scheduling heuristic	(StoL, MinET)			
Cloud resources to be composed by BAs	{1 standard–small, 1 standard–large, 1 standard–extra large, 1 micro, 1 high-memory–extra large, 1 high-memory–double extra large, 1 high-CPU–medium} all of them allocated for 10 h			
BoT execution request rate (requests per second)	Low		High	
	1		100	
Simulation time rate	60 s in simulation is equal to 1 h			

a set of Cloud resources to be composed by BAs where BoTs were executed, BoT execution request rates, and a simulation time rate.

The number of tasks per BoT (Table 7) was set from 50 to 300 tasks in 50-task increments to evaluate the performance of the agent-based concurrent scheduling with respect to different BoT size categories. Uniformly distributed BoTs were arbitrarily selected, given that how BoTs were generated is irrelevant to the evaluation of the agent-based concurrent BoT scheduling and execution approach.

In addition, all the randomly generated BoTs of a given BoT size category were the same, e.g., every time that an agent was dealing with a randomly generated 150-task BoT, the BoT contained the same tasks with the same number of instructions. In doing so, the agents were provided with scenarios where identical BoTs (submitted practically at the same, e.g., at a rate of 100 BoT submissions per second, see Table 8) were scheduled on identical sets of Cloud resources practically at the same time in a parallel manner. Then, by using the same scheduling heuristic, e.g., (StoL, MinET) heuristic, BAs, SPAs, and RAs were forced to carry out any interaction (resulted from the scheduling and execution of the BoTs) practically at the same time due to the fact that task completion times were the same for every BoT.

The number of CAs was set to 10, 20, and 30 to evaluate the agent-based scheduling approach using different levels of concurrent loads. Furthermore, the numbers of BAs and SPAs were both fixed to 5 to provide CAs and BAs with a sufficient set of BAs and heterogeneous SPAs to simulate the CNP and distributed BoT execution but handling multiple concurrent scheduling and execution of BoTs per BA and SPA.

Two levels of BoT execution request rates (i.e., number of BoT execution requests per second) were defined: *low* (1 request per second) and *high* (100 requests per second). The low request rate was based on the maximum number of requests per second accepted by some commercial web services [19] to evaluate the agent-based scheduling approach using real world settings. The high request rate was set to 100 requests per second to evaluate the agent-based scheduling approach in situations where the agent-based BoT execution system may be overloaded due to a high request rate of concurrent and parallel BoT executions.

The remaining experimental parameters (e.g., Cloud environment's input data, number of instructions per tasks, selection of the functional Cloud resource type, number of agents involved per CNP, simulation time rate, etc.) have the same justification presented in Section 5.1.

For each configuration of the agent-based testbed (Table 8) provided with the BoT-related input data (Table 7), 36 experiment runs were carried out, for an overall of 720 concurrently scheduled and executed BoTs.

(c) *Performance measures*. The performance measures are: BoT success rate, average BoT scheduling overhead time (amount of additional time, without counting task execution times, required by agents to prepare and schedule BoTs as well as time consumed by agents' reactions to messages and message latencies), and average number of messages exchanged for executing BoTs. All the performance measures were grouped by both BoT size category and BoT execution request rate. See Table 9 for details.

(d) *Results*. Experimental results are shown in Figs. 10 and 11. From these results, three observations were drawn.

Observation 1. Agents in the testbed were capable of successfully executing and scheduling BoTs concurrently with a 100% success rate for randomly generated BoTs of a wide variety of sizes in randomly generated Cloud environments even with high execution request rates and handling multiple BoTs at the same time.

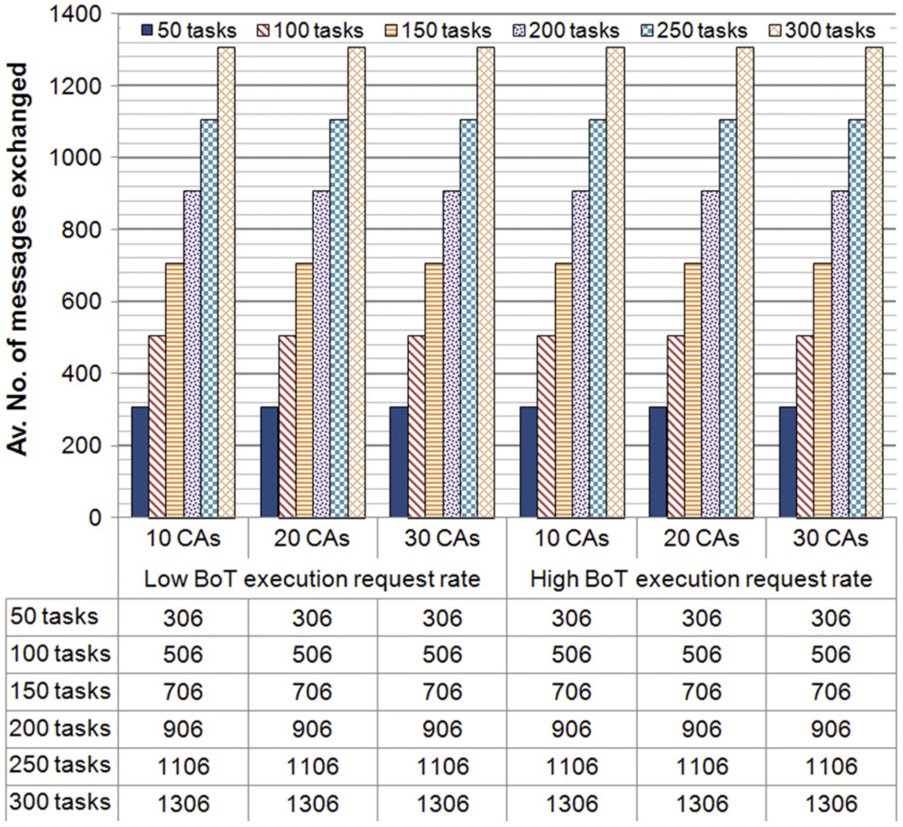
Analysis. The agents successfully executed BoTs because: (i) CAs were capable of finding BAs to submit BoTs for execution by adopting the CNP with BAs as participants. (ii) BAs successfully composed Cloud resources from a heterogeneous and distributed pool of SPAs by adopting the CNP in a parallel manner. (iii) BAs (using service contracts as a guide) requested the execution of tasks to the appropriate SPAs (i.e., SPAs previously contracted that contained the Cloud resource needed to execute a given task). (iv) BAs were capable of sorting BoT tasks and mapping them to the previously allocated Cloud resources based on the (StoL, MinET) scheduling heuristic. (v) SPAs effectively handled concurrent BAs' requests for task execution by forwarding the requests to their RAs in a parallel manner. (vi) RAs successfully executed all the tasks assigned, and reported their outputs to the SPAs, which forwarded them to the BAs.

Agents in the testbed achieved a 100% success rate in executing randomly generated BoTs concurrently and in a parallel manner even in the presence of high BoT execution request rates because through dynamic and flexible interaction, BAs, SPAs, and RAs effectively coordinated themselves to schedule and execute the BoTs.

Observation 2. Agents in the testbed efficiently scheduled and executed BoTs by sending only a linearly increasing number of messages among CAs, BAs, SPAs and RAs that depends only in the number of tasks contained in BoTs.

Table 9
Performance measures for Experiment 5.2.

Performance measure	Description
BoT success rate	N_{SUC}/N_{ATT}
Av. BoT scheduling overhead time	$\Sigma(T)/N_{ATT}$
Av. no. of messages exchanged for executing BoTs	$\Sigma(\mathrm{Msg})/N_{ATT}$
Performance measures' variables	**Description**
N_{SUC}	No. of successfully executed BoTs
Msg	Messages exchanged for executing a BoT
T	Overhead time for scheduling and executing a BoT
Performance measures' constant	**Possible values**
N_{ATT}: No. of attempts for BoT executions	{50, 100, 150, 200, 250, 300}



	Low BoT execution request rate			High BoT execution request rate		
	10 CAs	20 CAs	30 CAs	10 CAs	20 CAs	30 CAs
50 tasks	306	306	306	306	306	306
100 tasks	506	506	506	506	506	506
150 tasks	706	706	706	706	706	706
200 tasks	906	906	906	906	906	906
250 tasks	1106	1106	1106	1106	1106	1106
300 tasks	1306	1306	1306	1306	1306	1306

Fig. 10. Av. no. of messages exchanged for scheduling and executing BoTs concurrently.

Analysis. As shown in Fig. 10, on average, the number of messages exchanged among agents in the testbed increased linearly with the number of tasks to be executed. Agents sent messages for submitting BoTs for execution, composing the sets of Cloud resources, and coordinating the distributed and concurrent execution of BoTs, i.e., to submit tasks for execution and to forward tasks' outputs among BAs, SPAs, and RAs. However, most of the messages exchanged resulted from coordinating the execution of BoT tasks. Thus, more messages were exchanged when there were more tasks contained in the BoTs regardless of the BoT execution request rates and the number of BoTs executed concurrently. In addition, the number of messages increased linearly because for every message that a BA sent to an SPA for executing a task, the SPA also sent only one message to an RA for assigning the task, afterward, the RA also sent only one message to deliver the output to the SPA, which forwarded it to the BA.

In summary, agents endowed with distributed problem solving techniques through interaction can schedule and execute randomly generated BoTs in randomly generated Cloud environments both efficiently (in terms of messages exchanged) and effectively (see Section 5.2, Observation 1) in a concurrent manner.

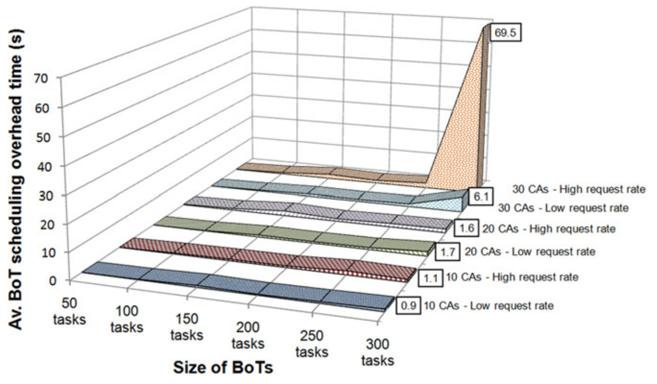


Fig. 11. Av. BoT scheduling overhead time of concurrently scheduled and executed BoTs.

Observation 3. For most of the cases, agents in the testbed efficiently scheduled and executed BoTs in a concurrent and parallel manner with relatively negligible BoT scheduling overhead times, i.e., overhead times that do not interfere with the effectiveness (see Section 5.2, Observation 1) of the scheduling and execution of BoTs. In addition, the overhead times were negligible even with high BoT execution request rates, concurrently scheduling, and executing from 10 to 30 randomly generated BoTs of a wide variety of sizes in distributed and randomly generated Cloud environments. In addition, in general, the BoT scheduling overhead time was higher with higher request rates, larger BoTs, and larger numbers of BoTs executed concurrently.

Analysis. As show in Fig. 11, for the majority of the cases, the BoT scheduling overhead time was (relatively) short for both low and high BoT execution request rates and the different levels of concurrency (10, 20, and 30 concurrently executed BoTs). This is because, for most of the cases, agents in the testbed were capable of handling the load resulted from concurrent execution of BoTs by distributing the load on different BAs and SPAs as a result of adopting the CNP for selecting among multiple agents, either BAs or SPAs to execute BoTs and allocate Cloud resources, respectively. In doing so, agents in the testbed were capable of promptly reacting to BoT task completions and sharply coordinating the distributed execution of BoTs, avoiding idle time of previously allocated Cloud resources.

As shown in Fig. 11, as more and larger BoTs were concurrently introduced into the system with higher request rates, the BoT scheduling overhead time was (in most of the cases) slightly increased due to the additional time consumed by: ordering tasks, adopting concurrent and parallel CNPs to allocate Cloud resources, distributing and assigning tasks to previously allocated Cloud resources, and exchanging messages to establish agent coordination and thus accumulating message latencies.

A special case was presented when agents in the testbed executed 300-task BoTs from 30 CAs concurrently with a high request rate (see Fig. 11), which had a considerably high (69.5 s) overhead time. This was due to the overloading of the Cloud directory (provided by a JADE system agent), which created a bottleneck when CAs and BAs were looking for BAs and SPAs to execute BoTs and compose Cloud resources, respectively. In addition, another cause for such overhead time was the saturation of the internal message queues of agents, given that CAs submitted identical 300-task BoTs that forced BAs, SPAs, and RAs to carry out the interaction resulted from the scheduling and execution of the 30 BoTs practically at the same time and thus saturating agents' message queues. Nonetheless, agents in the testbed executing 300-task BoTs from 30 CAs concurrently with a low request rate (see Fig. 11) had a much shorter overhead time: 6.1 s due to lesser exposure to the saturation of agents' message queues and the bottleneck of the Cloud directory.

In summary, agents' distributed and cooperative problem solving techniques are efficient enough to support concurrent scheduling and execution of multiple BoTs with extremely high execution request rates and stringent constraints (fewer BAs and SPAs, and concentrating agent interaction in short periods of time by scheduling and executing identical BoTs) and yet, in most of the cases, getting negligible BoT scheduling overhead times. Moreover, even in the worst cases (e.g., 69.5 s of overhead time), the overhead time was relatively negligible compared to the execution time required by BoTs executed in Cloud environments.

5.3. Evaluating the agent-based BoT scheduling endowed with I-ElasticAM

(a) *Objective*. A series of experiments was designed to evaluate the efficiency of the agent-based Cloud BoT concurrent scheduling approach endowed with I-ElasticAM.

(b) *Benchmark Cloud resource allocation mechanisms*. Two Cloud resource allocation mechanisms were used for benchmarking the proposed I-ElasticAM:

- *Fixed Cloud resource allocation mechanism* (FixedAM) consisting of allocating in advance all the Cloud resources for a fixed number of hours to execute a given BoT. This resource allocation mechanism requires a fixed number of hours to execute BoTs and lacks elasticity.
- *Generalized elastic Cloud resource allocation mechanism* (GeneralizedAM) consisting of allocating all the Cloud resources designated to execute a given BoT in an hourly basis until the BoT is completely executed, i.e., the status of the BoT execution is verified every hour, if it is not completely executed, all the Cloud resources are reallocated for the next hour. A similar allocation mechanism can be found in [6].

Section 5.3 includes a quantitative comparison between the benchmark Cloud resource allocation mechanisms and the proposed I-ElasticAM, see Section 6 for a qualitative comparison.

(c) *Experimental settings*. The testbed has three different sets of input parameters: (1) Cloud environment's input data (Table 3), (2) BoT-related input data (Table 10), (3) Agent-based testbed parameters (Table 11). Cloud environment's input data consists of nonfunctional types of Cloud resources described by their computing capacities and hourly cost rates. BoT-related input data consists of number of tasks per BoT, number of instructions per task, probability distributions used for creating randomly generated BoTs, and a functional Cloud resource type (e.g., a rendering service). Agent-based testbed parameters consist of number of agents involved in the simulation grouped by agent type, number of agents involved per CNP execution, Cloud scheduling heuristics, a set of Cloud resources to be composed by BAs where BoTs were executed, the Cloud resource allocation time for FixedAM, the reallocation threshold for I-ElasticAM, and a simulation time rate.

Since 300-task BoTs take longer to be executed, their execution makes more use of the elastic Cloud resource allocation mechanisms, and thus highlight the performance of the Cloud resource allocation mechanisms evaluated. In addition, experiment runs using the different types (uniformly, normally, right-half-normally, and left-half-normally distributed) of randomly generated BoTs were carried out given that the execution time of BoTs varies among them, and thus different levels of elasticity (i.e., allocation of more or less hours of a given Cloud resource) are required. The remaining BoT-related input data parameters have the same justifications provided in Section 5.1.

The agent-based testbed parameters were based on the design of the experiment of Section 5.1. However, in this case, CAs submitted BoTs in a concurrent manner to show the compatibility of the elastic Cloud resource allocation mechanisms with the concurrent handling of the agent-based scheduling approach. All the proposed Cloud scheduling heuristics (Table 11) were used in order to test the capability of I-ElasticAM to work even when no information about task completion times is available. In addition, three allocation mechanisms were used: I-ElasticAM and the two benchmark resource allocation mechanisms (GeneralizedAM and FixedAM). The resource allocation time for FixedAM was set to 10 h for its (previously determined) sufficiency for executing the BoTs. The reallocation threshold (see Section 3) for I-ElasticAM was set to 1 min to reduce the probability of unnecessary reallocation of Cloud resources when no information about task completion times is given, but at the same time provide BAs and SPAs with sufficient time to reallocate Cloud resources.

For each configuration of the agent-based testbed (Table 11) provided with the BoT-related input data (Table 10), 168 experiment runs were carried out, for an overall of 840 concurrently scheduled and executed BoTs.

Table 10
BoT-related input data source for Experiment 5.3.

Input data	Possible values
BoT size (No. of tasks per BoT)	300 tasks
Task size (No. of instructions per task)	{400, 800, 1200, 1600, 2000, 2400, 2800, 3200, 3600, 4000} million of instructions
Probability distribution used for creating randomly generated BoTs	{*uniform*, *normal*, *left-half-normal*, *right-half-normal*}
Functional Cloud resource type	Homogeneous: $\{s_1\}$

Table 11
Agent-based testbed parameters for Experiment 5.3.

Input data	Possible values			
Agent types	CAs	BAs	SPAs	RAs
No. of agents	5	5	5	2500
Agents involved per CNP	1 manager and 3 contractors			
Cloud scheduling heuristics	{(U, MaxET), (U, MaxCT), (U, MinET), (U, MinCT), (LtoS, R) (LtoS, MaxET), (LtoS, MaxCT), (LtoS, MinET), (LtoS, MinCT), (StoL, R), (StoL, MaxET), (StoL, MaxCT), (StoL, MinET), (StoL, MinCT)}			
Cloud resources to be composed by BAs	{1 standard–small, 1 standard–large, 1 standard–extra large, 1 micro, 1 high-memory–extra large, 1 high-memory–double extra large, 1 high-CPU–medium}			
Allocation mechanisms	(1) {I-ElasticAM, GeneralizedAM, FixedAM}			
Allocation time for FixedAM	10 h			
Reallocation threshold for I-ElasticAM	1 min			
Simulation time rate	60 s in simulation is equal to 1 h			

Table 12
Performance measures for Experiment 5.3.

Performance measures	Description
Av. BoT makespan	$\Sigma(M)/N$
Av. no. of messages exchanged	$\Sigma(\text{Msg})/N$
Av. BoT execution cost	$\Sigma(C)/N$
Performance measures' variables	**Description**
M	Makespan of a BoT executed using a given scheduling heuristic grouped by *BoT type*.
Msg	Messages exchanged for executing a given BoT grouped by *BoT type*.
C	Cost of executing a BoT in the Cloud grouped by *BoT type*.
N	No. of BoTs executed using a given scheduling heuristic grouped by *BoT type*.
Performance measures' constant	**Possible values**
BoT types	{uniformly, normally, right-half-normally, and left-half-normally} distributed BoTs.

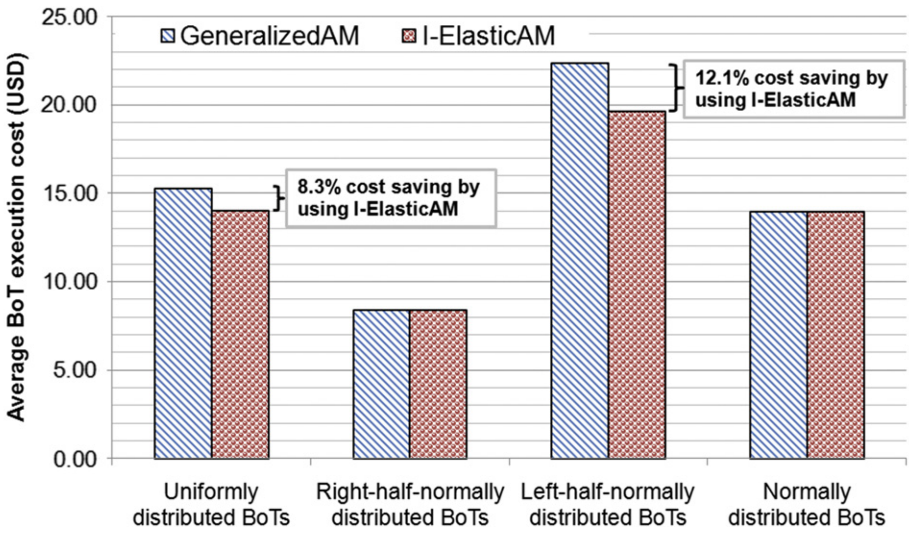


Fig. 12. Av. BoT execution cost—GeneralizedAM versus I-ElasticAM.

(d) *Performance measures*. The performance measures are: average BoT makespan, average number of messages exchanged, and average BoT execution cost. See Table 12 for details.

(e) *Results*. Experimental results are shown in Figs. 12–14. From these results, two observations were drawn.

Observation 1. Agents in the testbed scheduling and executing BoTs using I-ElasticAM attained lower (or at least the same) BoT execution costs than agents using GeneralizedAM.

Analysis. As shown in Fig. 12, in 2 out of 4 cases, agents in the testbed using I-ElasticAM scheduled and executed BoTs with

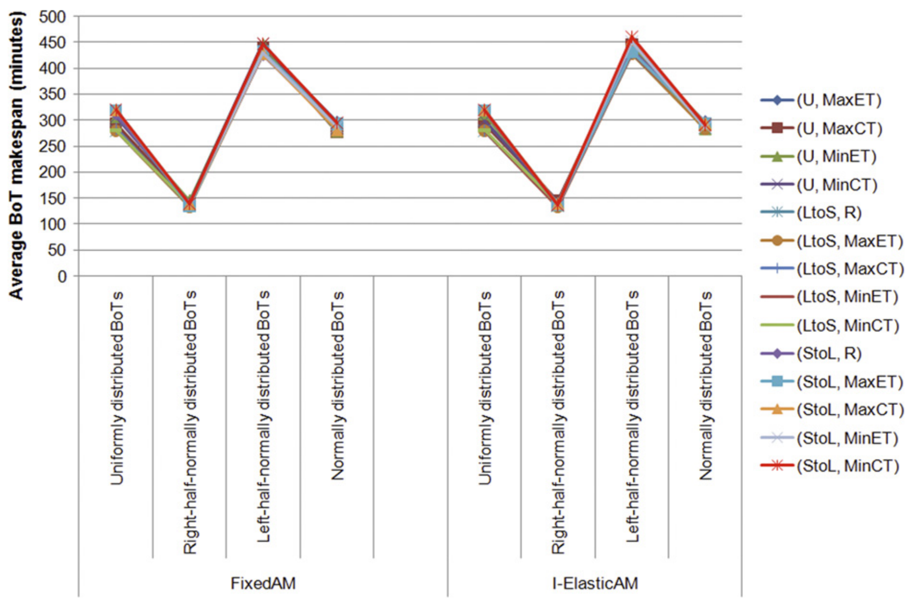


Fig. 13. Av. BoT makespan—FixedAM versus I-ElasticAM.

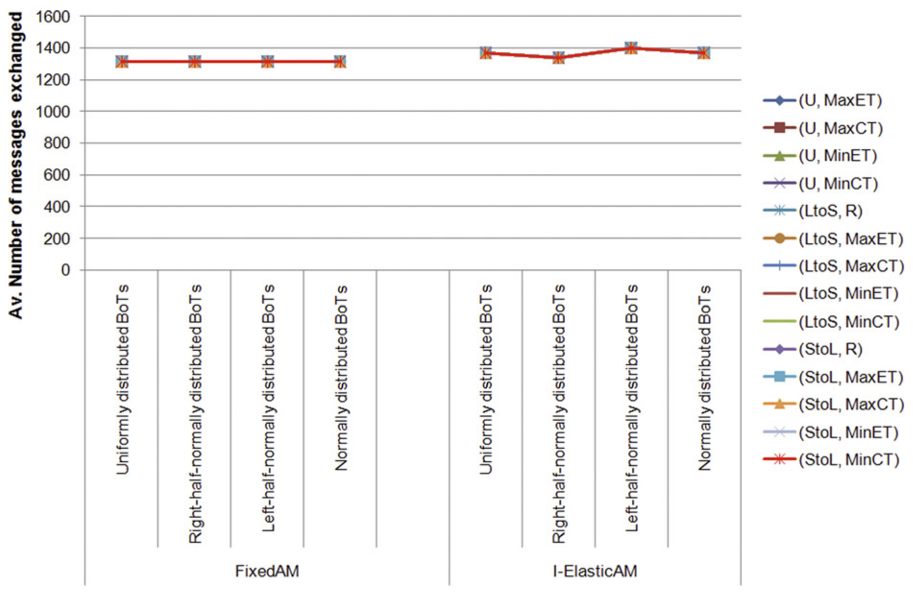


Fig. 14. Av. no. of messages exchanged—FixedAM versus I-ElasticAM.

lower execution cost (attaining up to 8.3% and 12.1% execution cost savings) than agents using GeneralizedAM. This is because behavior I-ElasticAM_BA (see Section 3) built into BAs monitored the status of each Cloud resource allocated, and then whenever BAs detected idle Cloud resources just before exhausting their allocation time (determined by the reallocation threshold), the Cloud resources were not reallocated for the next 1-hour time slot regardless of whether the BoTs had been completely executed. In doing so, only the Cloud resources that were still executing BoT tasks were reallocated for the next 1-hour time slot. Conversely, agents using GeneralizedAM based on the completion of BoTs, blindly reallocated all the (idle and busy) Cloud resources for the next 1-hour time slot increasing the BoT execution costs.

As shown in Fig. 12, in 2 out of 4 cases, agents in the testbed using I-ElasticAM scheduled and executed BoTs with the same execution cost as agents using GeneralizedAM. This is because the Cloud resources allocated for the execution of the right-half-normally distributed BoTs (Fig. 12, 2nd column) and the normally distributed BoTs (Fig. 12, 4th column) completed the execution of their corresponding BoT tasks in the same 1-hour time slot, as shown in Fig. 12 in the 2nd and 4th columns. This indicates that the Cloud resources allocated for executing the right-half-normally and normally distributed BoTs completed their execution in less than 3 (see Fig. 13, 2nd column) and 5 h (see Fig. 13, 4th column) spending $8.38 USD (see Fig. 12, 2nd column) and 13.97 USD (see Fig. 12, 4th column), respectively.

By making use of their parallel working capabilities, agents in the testbed can (at the same time that concurrently schedule and execute BoTs) monitor individual Cloud resource remaining allocation times and the individual status (e.g., idle) of Cloud resources supported by the existence of service level agreements of Clouds, conforming to I-ElasticAM that autonomously reallocates Cloud resources as needed by BoTs even without information of task completion times and Cloud resources' computing capacities.

Observation 2. Agents adopting I-ElasticAM achieved the same performance (regarding BoT makespan) as agents adopting FixedAM. In addition, agents adopting I-ElasticAM exchanged only very few additional messages to autonomously and elastically reallocate Cloud resources to schedule and execute BoTs.

Analysis. As shown in Fig. 13, the average BoT makespan achieved by agents adopting I-ElasticAM and by agents adopting FixedAM was very similar for all the different BoT types and for all the Cloud scheduling heuristics. This is because I-ElasticAM was built into agents as a parallel behavior (i.e., agent functionality) for each Cloud resource allocated, and such parallel agent behaviors were cyclically activated by a timer determined by a reallocation threshold (see Section 3) and the length of the allocation slots (e.g., 1 h). Thus, due to its parallel and event-oriented nature, I-ElasticAM did not alter the BoT makespans.

As shown in Fig. 14, the number of messages exchanged by agents adopting I-ElasticAM was slightly larger than the number of messages exchanged by agents adopting FixedAM. As previously stated in Observation 2 of Section 5.2, the number of messages exchanged among agents mostly depends on the number of tasks contained in BoTs (see the number of messages exchanged by agents adopting FixedAM in Fig. 14). However, by including I-ElasticAM into the agent-based scheduling BoT approach, BAs sent one message to SPAs to request the reallocation of a given Cloud resource for the next time slot, in reply, SPAs sent one *acknowledgment* message to BAs. So, the overall number of messages exchanged mostly depends on the size of BoTs, but also (although in a much lesser extent) on the BoT makespan, given that the longer it takes to execute BoTs, the more messages are exchanged among agents to coordinate the elastic Cloud resource reallocation.

In summary, agents adopting I-ElasticAM are capable of autonomously and dynamically reallocating Cloud resources to execute BoTs efficiently (by exchanging just a few messages more and maintaining the same BoT makespan as agents adopting FixedAM), in addition to reducing BoT execution costs.

6. Related work comparison

BoT scheduling heuristics are commonly focused on minimizing makespan [20,11,15,4,8,17] and use information ranging from computing resources' ready times [17], to earliest completion times of tasks [11,4]. In addition, BoT scheduling approaches that do not require any information about tasks or computing resources have been designed and are commonly based on tasks' replications. Replication-based scheduling approaches [20,21,15,8,22] map tasks onto computing resources in an unordered manner to any unoccupied computing resource, but whenever all the tasks are currently in execution and computing resources become available, unfinished tasks are replicated in the hope that such resources can complete the tasks before the originally initiated tasks. Moreover, once replicated tasks are completely executed, all its replicas are canceled to execute the remaining tasks. In this regard, replication-based scheduling approaches assume that tasks are preemptive, i.e., task execution can be interrupted at any moment.

Most of the BoT scheduling heuristics are designed under the assumption that BoT holders are only charged for task execution time or not charged at all, e.g., BoT execution in Grid environments [20,11,15,8,17,23]. In contrast, in Cloud environments, BoT holders are charged for complete (1-hour) allocation slots of computing resources [1,24], disregarding whether tasks are being executed.

In the area of elastic Cloud resource allocation mechanisms, the research effort presented in [6] addressed BoT scheduling using an immediate mode scheduling heuristic to schedule BoTs in hybrid environments (i.e., using both Grid and Cloud computing resources). In [6], Cloud resources are reallocated every hour (as in GeneralizedAM, see Section 5.3, incise b) based on an estimation of the system throughput to determine whether a given BoT will be completely executed in the next time slot. Nonetheless, [6] assumes that all the Cloud resources are identical and have constant performances. Other approaches that take advantage of the elasticity of Clouds are focused on dynamically adding or removing Cloud resources during the execution of consumers' applications, such as [25] in Cloud environments and [26] in Grid environments.

In this work, tasks are assumed to be non-preemptive, i.e., once task execution is started, it cannot be interrupted, unlike replication-based approaches [20,21,8,22]. In addition, given that Cloud resources' allocation time is running regardless of whether tasks are being executed (unlike Grid environments where users are normally charged for task execution time or not charged at all [20,11,15,8,17]), maximizing the use of Cloud resources results in an efficient use of consumers' budgets.

Unlike previous Cloud-oriented BoT scheduling and execution approaches [6,22,27,28], this work makes use of the CNP to dynamically select and compose the best (cheapest) available Cloud resources from a pool of heterogeneous Cloud providers to execute BoTs in a distributed manner.

In the area of elastic Cloud resource allocation mechanisms, I-ElasticAM (in contrast to [6]) only requires a predefined initial set of Cloud resources, and no estimation of Cloud resources' throughput or constant performance of Cloud resources is required. Moreover, I-ElasticAM can work with Cloud resources allocated at different times due to its individualized monitoring and reallocation mechanism, so new Cloud resources can be dynamically added. Furthermore, I-ElasticAM can complement (rather than replace) approaches that dynamically add computing resources during the execution of consumers' applications [29,30,25,26,31].

In quantitative terms, executing BoTs using (some of) the proposed Cloud scheduling heuristics outperformed (in some cases) the benchmark scheduling heuristics (Max–min, Min–min [11,4], FCFS, Greedy-P, Greedy-R [16]). Nevertheless, in some cases, some benchmark scheduling heuristics (e.g., Max–min) outperformed the proposed scheduling heuristics. However, the support provided by the remaining allocation time of Cloud resources allows dynamic inclusion of Cloud resources during BoT execution without rescheduling (as Min–min, Max–min, Greedy-P, and Greedy-R would need) because of the decoupling of the task mapping policies from the task orderings.

It is acknowledged, that this work is a considerably and significantly extended version of a preliminary work reported in [24]. The present work has augmented [24] by devising, implementing, and evaluating an elastic Cloud resource allocation mechanism: I-ElasticAM (Sections 3 and 5.3). In addition, the agents were endowed with parallel working capabilities capable of scheduling and executing BoTs in a concurrent and parallel manner (Sections 2 and 5.2). Moreover, empirical results obtained in [24] were generalized by means of conducting new experiments in a major scale using multiple probability distributions to randomly generate BoTs. In doing so, the efficiency of the 14 Cloud scheduling heuristics was explored in a multitude of situations that resulted in a clear identification of favorable and unfavorable scenarios for each Cloud scheduling heuristic (Section 5.1). Additionally, the time complexity of the entire family of the proposed Cloud scheduling heuristics was identified (Section 4). Furthermore, two greedy batch mode scheduling heuristics: Greedy-P and Greedy-R (Section 5.1), and the GeneralizedAM elastic allocation mechanism (Section 5.3) were implemented for benchmarking purposes. Finally, the analysis and scope of the related work were extended (Section 6).

In addition, it is acknowledged that only the agent-based approach for composing Cloud resources was based on [32].

7. Conclusion and future work

The novelty of this work is that, to the best of the authors' knowledge, it is among the earliest works to adopt an agent-based approach for concurrent Cloud BoT scheduling using a family of 14 Cloud scheduling heuristics as well as dealing with elastic Cloud resource reallocation. Whereas [2] has proven that the scheduling of independent tasks in a set of heterogeneous computing resources is an NP-complete problem, the significance of this work is devising a family of 14 scheduling heuristics, from which three scheduling heuristics are shown to be in linearithmic time (R-based scheduling heuristics: $\{\text{LtoS}, \text{StoL}\} \times \{\text{R}\}$) and the rest in polynomial time.

The contributions of this work are detailed as follows:

(1) Providing a family of 14 Cloud scheduling heuristics (Section 4) adapted to the resource allocation settings (e.g., 1-hour time slots in Amazon EC2 [1]) of Clouds based on the remaining allocation time of Cloud resources, which is always available due to the existence of service level agreements established between consumers and Cloud providers. By using the remaining allocation of Cloud resources, the utilization of Cloud resources that are about to expire is prioritized to use as much as possible the computing resources already allocated and paid. In addition, the family of Cloud scheduling heuristics supports the dynamic inclusion of new Cloud resources while scheduling and executing a given BoT without rescheduling because of the decoupling of the task mapping policies from the task orderings.

(2) Devising an elastic Cloud resource allocation mechanism: I-ElasticAM (Section 3) that autonomously and dynamically reallocates Cloud resources on demand to BoT executions even without information of task completion times and Cloud resource computing capacities. I-ElasticAM along with the proposed Cloud scheduling heuristics take advantage of the elasticity of Cloud environments to allocate Cloud resources only as much as needed to execute a given BoT in a dynamic and autonomous manner while making an efficient use of consumers' budgets.

(3) Implementing an agent-based Cloud BoT scheduling approach (Section 2) that supports concurrent and parallel scheduling and execution of BoTs, and concurrent and parallel dynamic selection and composition of Cloud resources (by using the CNP) from multiple and distributed Cloud providers. By scheduling and executing simultaneously submitted BoTs in Cloud environments in a concurrent and parallel manner, brokers and/or providers can handle heavy workloads when enough Cloud resources are available. In addition, by adopting the CNP, the agents are provided with a mechanism for agreeing fees for executing BoTs and allocating Cloud resources, which supports Cloud participants' autonomy and self-interestedness.

(4) Providing experimental evidence to demonstrate the efficiency of the Cloud scheduling heuristics (Section 5.1), the effectiveness of the agent-based Cloud BoT concurrent scheduling and execution approach (Section 5.2), and the efficiency of the elastic Cloud resource reallocation mechanism (Section 5.3). From the empirical results, it can be concluded that the agents can schedule and execute BoTs in a concurrent manner in distributed Cloud environments as follows:
   - Efficiently executing BoTs by attaining similar (in some cases shorter) makespans to commonly used benchmark heuristics such as FCFS, Min–min, Max–min, Greedy-R, and Greedy-P (Section 5.1).
   - Effectively executing BoTs by achieving a 100% success execution rate (Section 5.2).
   - Efficiently using Cloud consumers' budgets by using I-ElasticAM to elastically reallocate Cloud resources on demand as needed by BoT executions (Section 5.3).

In addition, agents in the testbed can efficiently schedule and execute BoTs by sending only a linearly increasing number of messages and attaining negligible overhead times even with high BoT execution request rates and with different numbers of BoTs executed in a concurrent and parallel manner (Section 5.2).

This work has demonstrated that Cloud scheduling heuristics based on Cloud resources' remaining allocation times are efficient and suitable for executing BoTs in Cloud environments. In addition, the proposed agent-based approach for concurrent and parallel scheduling and execution of BoTs in Cloud environments has been empirically shown to be effective and efficient. Moreover, this work has demonstrated that the agents can autonomously and dynamically reallocate Cloud resources on demand to elastically execute BoTs in multiple Cloud environments. Furthermore, this work has demonstrated that the Cloud resources' remaining allocation time can support individualized elastic resource reallocation as needed by BoT executions autonomously.

It should be acknowledged that, whereas in this work, scheduling heuristics that require no information of either Cloud resources or tasks are proposed (e.g., (U, MaxCT) and (U, MinCT) heuristics), some of the proposed Cloud scheduling heuristics have a dependency on knowledge about the computing capacity of Cloud resources (e.g., MaxET-based and MinET-based heuristics) and/or dependency on knowledge about task size (e.g., LtoS-based and StoL-based heuristics). Thus, a mechanism to relax such assumptions on knowledge about Cloud resources and tasks should be provided by including an initial task profiling phase, such as in [22,33]. In addition, the proposed Cloud scheduling heuristics are focused on compute-intensive BoTs [8], i.e., BoTs that mostly require computing resources (e.g., powerful processors) to be executed, and thus overhead times associated to data transfer and data storage are disregarded. However, when dealing with data-intensive BoTs [8], i.e., BoTs handling and transferring considerably large amounts of data, the overhead times associated to data transfers should be taken into account, such as in [34] to efficiently schedule and execute data-intensive BoTs.

Future research directions are as follows. Relaxing assumptions on knowledge about Cloud resources and tasks by implementing an initial task profiling phase. Designing Cloud scheduling heuristics for handling data-intensive BoTs. Devising a Cloud resource estimation method for determining an initial set of Cloud resources to be allocated for the initial hour to execute BoTs based on consumers' budgets and deadlines. Deploying the agent-based elastic Cloud BoT concurrent scheduling approach in commercial Clouds (e.g., Amazon EC2). Carrying out experiments in a much larger scale in commercial Clouds to evaluate the scalability of the agent-based BoT scheduling approach. Designing agent-based distributed problem solving techniques to handle BoT scheduling and execution in unreliable Cloud environments.

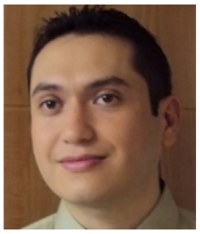
Acknowledgments

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MEST) (KRF-2009-220-D00092) and the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea Government (MEST 2009-0065329). The authors would like to thank the Guest Editors and the anonymous referees for their comments and suggestions.

References

[1] Amazon EC2 http://aws.amazon.com/ec2/, 2011 (accessed 10.06.11).
[2] J. Bruno, E.G. Coffman Jr., R. Sethi, Scheduling independent tasks to reduce mean finishing time, Communications of the ACM 17 (7) (1974) 382–387.
[3] E. Horowitz, S. Sahni, Exact and approximate algorithms for scheduling nonidentical processors, Journal of the ACM 23 (2) (1976) 317–327.

[4] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on nonidentical processors, Journal of the ACM 24 (2) (1977) 280–289.
[5] C. Anglano, M. Canonico, Scheduling algorithms for multiple bag-of-task applications on desktop grids: a knowledge-free approach, in: Proc. IEEE International Symposium on Parallel and Distributed Processing, IEEE Press, 2008, pp. 1–8.
[6] D. Candeia, R. Araujo, R. Lopes, F. Brasileiro, Investigating business-driven cloudburst schedulers for E-science bag-of-tasks applications, in: Proc. IEEE 2nd International Conference on Cloud Computing Technology and Science, IEEE Press, 2010, pp. 343–350.
[7] Y.-H. Lee, S. Leu, R.-S. Chang, Improving job scheduling algorithms in a grid environment, Future Generation Computer Systems 27 (8) (2011) 991–998.
[8] Y.C. Lee, A.Y. Zomaya, Practical scheduling of bag-of-tasks applications on grids with dynamic resilience, IEEE Transactions on Computers 56 (6) (2007) 815–825.
[9] M. Silberstein, A. Sharov, D. Geiger, A. Schuster, Gridbot: execution of bags of tasks in multiple grids, in: Proc. Conference on High Performance Computing Networking, Storage and Analysis, ACM, New York, 2009, p. 12. Article 11.
[10] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems, in: Proc. of the 8th Heterogeneous Computing Workshop, IEEE Computer Society, Washington, DC, USA, 1999, pp. 30–44.
[11] R.F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J.D. Lima, F. Mirabile, L. Moore, B. Rust, H.J. Siegel, Scheduling resources in multi-user, heterogeneous, computing environments with smartnet, in: Proc. 7th Heterogeneous Computing Workshop, IEEE Press, 1998, pp. 3–19.
[12] R.G. Smith, The contract net protocol: high-level communication and control in a distributed problem solver, IEEE Transactions on Computers 29 (12) (1980) 1104–1113.
[13] C.H. Papadimitriou, Computational complexity, in: A. Ralston, E.D. Reilly, D. Hemmendinger (Eds.), Encyclopedia of Computer Science, fourth ed., John Wiley and Sons Ltd., Chichester, UK, 2003, pp. 260–265.
[14] F. Bellifemine, A. Poggi, G. Rimassa, JADE-A FIPA-compliant agent framework, in: Proc. 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, 1999, pp. 97–108.
[15] N. Fujimoto, K. Hagihara, A comparison among grid scheduling algorithms for independent coarse-grained tasks, in: Proc. 2004 Symposium on Applications and the Internet- Workshops, IEEE Press, 2004, pp. 674–680.
[16] K. Li, Job scheduling and processor allocation for grid computing on metacomputers, Journal of Parallel and Distributed Computing 65 (11) (2005) 1406–1418.
[17] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, Journal of Parallel and Distributed Computing 59 (2) (1999) 107–131.
[18] B.R. Moulton, Bias in the consumer price index: what is the evidence? Journal of Economic Perspectives, American Economic Association 10 (4) (1996) 159–177.
[19] Amazon Product Advertising API License Agreement https://affiliate-program.amazon.com/gp/advertising/api/detail/agreement.html, 2011 (accessed 10.06.11).
[20] W. Cirne, D.P. da Silva, L. Costa, E. Santos-Neto, F.V. Brasileiro, J.P. Sauve, F.A.B. Silva, C.O. Barros, C. Silveira, Running bag-of-tasks applications on computational grids: the mygrid approach, in: Proc. International Conference on Parallel Processing, 2003, pp. 407–416.
[21] N. Fujimoto, K. Hagihara, Near-optimal dynamic task scheduling of independent coarse-grained tasks onto a computational grid, in: Proc. International Conference on Parallel Processing, 2003, pp. 391–398.
[22] A.M. Oprescu, T. Kielmann, Bag-of-Tasks scheduling under budget constraints, in: Proc. IEEE 2nd International Conference on Cloud Computing Technology and Science, IEEE Press, 2010, pp. 351–359.
[23] J. Wu, X. Xu, P. Zhang, C. Liu, A novel multi-agent reinforcement learning approach for job scheduling in grid computing, Future Generation Computer Systems 27 (5) (2011) 430–439.
[24] J.O. Gutierrez-Garcia, K.M. Sim, A family of heuristics for agent-based Cloud bag-of-tasks scheduling, in: Proc. 3rd International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, IEEE Press, 2011, pp. 416–423.
[25] S. Pandey, D. Karunamoorthy, R. Buyya, Workflow engine for clouds, in: R. Buyya, J. Broberg, A. Goscinski (Eds.), Cloud Computing: Principles and Paradigms, Wiley Press, New York, USA, 2011, pp. 321–344.
[26] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde, Falkon: a fast and light-weight task execution framework, in: Proc. of the 2007 ACM/IEEE Conference on Supercomputing, ACM, New York, NY, USA, 2007, p. 12. Article 43.
[27] M.A. Salehi, R. Buyya, Adapting market-oriented scheduling policies for Cloud computing, in: C.-H. Hsu, et al. (Eds.), ICA3PP 2010, Part I, in: LNCS, vol. 6081, Springer, Heidelberg, 2010, pp. 351–362.
[28] J.N. Silva, L. Veiga, P. Ferreira, Heuristic for resources allocation on utility computing infrastructures, in: Proc. 6th International Workshop on Middleware for Grid Computing, ACM, New York, 2008, pp. 9–17.
[29] E.-K. Byun, Y.-S. Kee, J.-S. Kim, S. Maeng, Cost optimized provisioning of elastic resources for application workflows, Future Generation Computer Systems 27 (8) (2011) 1011–1026.
[30] S. Islam, J. Keung, K. Lee, A. Liu, Empirical prediction models for adaptive resource provisioning in the cloud, Future Generation Computer Systems 28 (1) (2012) 155–162.
[31] C. Vecchiola, R.N. Calheiros, D. Karunamoorthy, R. Buyya, Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka, Future Generation Computer Systems 28 (1) (2012) 58–65.
[32] J.O. Gutierrez-Garcia, K.M. Sim, Agent-based service composition in cloud computing, in: T.H. Kim, et al. (Eds.), GDC/CA 2010, in: CCIS, vol. 121, Springer, Heidelberg, 2010, pp. 1–10.
[33] S. Seneviratne, D.C. Levy, Task profiling model for load profile prediction, Future Generation Computer Systems 27 (3) (2011) 245–255.
[34] C. Briquet, X. Dalem, S. Jodogne, P.-A. de Marneffe, Scheduling data-intensive bags of tasks in P2P grids with bittorrent-enabled data distribution, in: Proc. 2nd Workshop on Use of P2P, GRID and Agents for the Development of Content Networks, ACM, New York, NY, USA, 2007, pp. 39–48.



J. Octavio Gutierrez-Garcia is a postdoctoral fellow at Multiagent and Cloud Computing Systems Laboratory, Gwangju Institute of Science and Technology, South Korea. He received his Ph.D. in Electrical Engineering and Computer Science from CINVESTAV (Mexico) and Grenoble Institute of Technology (France), respectively. Dr. Gutierrez-Garcia has served as a reviewer for various international conferences and as a member of editorial boards of scientific journals, including the Multiagent and Grid Systems—an International Journal (IOS press). He has conducted research on Cloud computing, distributed artificial intelligence, multiagent systems, and service computing.

Kwang Mong Sim is a Chair (Professor) in Computer Science at the University of Kent, Chatham Maritime, Kent, UK. Professor Sim was the Director of the Multiagent and Cloud Computing Systems Laboratory, Gwangju Institute of Science and Technology, South Korea. Prof. Sim is an Associate Editor of the IEEE Transactions on Systems, Man and Cybernetics—Part C and serves in the editorial boards of numerous international journals, including the International Journal of Cloud Computing (Inderscience). He is also the Guest Editor of five journal special issues in agent-based Grid computing, including IEEE Systems Journal's special issue on Grid resource management. He was a referee for the National Science Foundation, USA and a keynote speaker in numerous international conferences.