The Computer Journal Advance Access published January 9, 2015

doi:10.1093/comjnl/bxu158

WaaS: Workflow-as-a-Service for the Cloud with Scheduling of Continuous and Data-Intensive Workflows

Sérgio Esteves* and Luís Veiga

INESC-ID Lisboa, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal
**Corresponding author: sesteves@gsd.inesc-id.pt*

Data-intensive and long-lasting applications running in the form of workflows are being increasingly dispatched to cloud computing systems. Current scheduling approaches for graphs of dependencies fail to deliver high resource efficiency while keeping computation costs low, especially for continuous data processing workflows, where the scheduler does not perform any reasoning about the impact new input data may have in the workflow final output. To face such a challenge, we introduce a new scheduling criterion, Quality-of-Data (QoD), which describes the requirements about the data that are worthy of the triggering of tasks in workflows. Based on the QoD notion, we propose a novel service-oriented scheduler planner, for continuous data processing workflows, that is capable of enforcing QoD constraints and guide the scheduling to attain resource efficiency, overall controlled performance and task prioritization. To contrast the advantages of our scheduling model against others, we developed WaaS (Workflow-as-a-Service), a workflow coordinator system for the Cloud where data is shared among tasks via cloud columnar database.

Keywords: data processing workflow; data-intensive; scheduling; continuous processing; cloud computing; quality-of-service

Received 25 January 2014; revised 13 October 2014
Handling editor: Surya Nepal

1. INTRODUCTION

Data-intensive applications generally comprehend several distinct and inter-connected processing steps that can be expressed through a directed acyclic graph (DAG) and viewed as a workflow applying various transformations on the data. Such applications have been used in a large number of fields, e.g. assessing the level of pollution in a given city [1], detecting gravitational-waves [2], weather forecasting [3], predicting earthquakes [4], among others. The computation of such applications are being increasingly more dispatched to the Cloud, taking advantage of the utility computing paradigm. In this environment, scheduling plays a crucial role on delivering high performance, resource utilization and efficiency, while still meeting budget constraints.

Scheduling algorithms for workflows in the Cloud usually try either to minimize the overall completion time (or makespan) given a fixed budget, or to minimize the cost given a deadline. In workflows for continuous processing, resources are often wasted due to the small impact that data given as new input might have. This happens specially in monitoring activities, e.g. fire risk, air pollution, observing near-earth objects. Moreover, Workflow Management Systems (WMSs) typically disregard any semantics with respect to the output data, that could be used to reason about the amount of re-executions needed for a given data to be processed. As data may not always have the same impact and significance, we introduce a new scheduling constraint, named Quality-of-Data (QoD).

QoD[1] describes the minimum impact that new input data needs to have in order to trigger re-execution of processing steps in a workflow. This impact is measured in terms of data size, magnitude of values and update frequency. Having the QoD notion, we are thus able to change the workflow triggering semantics to be guided by the volume and importance that data communicated between processing steps might have on

[1] *QoD* is akin to Quality-of-Service, and should not be confused with issues such as internal data correctness, semantic coherence, data adherence to real-life sources, or data appropriateness for managerial and business decisions.

causing significant and meaningful changes in the values of final output steps. QoD can also be seen as a metric of triggering relaxation or optimist reusage of previous results.

From the user (or consumer) point of view, reducing costs while meeting a deadline is what matters most. In turn, cloud providers are interested in having low prices and making resource utilization as efficient as possible. This volition on both sides gains a special importance for long-running tasks, where intelligent SLAs may come into place. These SLAs can be seen as QoD constraints that allow cloud providers to give lower costs in exchange of some relaxation.

By allowing QoD-based relaxation, cloud services providing workflow execution (on a pay-per-execution basis) can define different service-level agreements (SLA) with lower prices. With cloud consumers specifying QoD constraints for each task, a WMS would be able to offer reduced prices due to resource savings, and still give the best possible quality within the QoD to normal-execution range.

Having the current outlook, we propose the use of a novel workflow model and introduce a new scheduling algorithm for the Cloud that is guided by QoD, budget, and time constraints. We also present the design of WaaS (Workflow-as-a-Service), a WMS platform that portrays our vision of a Cloud service offered at the PaaS level, on top of virtualization technology and the HBase [5] noSQL storage, bridging the gap between traditional WMS and utility computing. Results show that we are able to reduce costs by the use of our QoD model.

Without a Cloud service as WaaS, users would need not only to possess the knowledge on how to setup a WMS (which is non-trivial), but would also have to spend substantial amounts of effort in making decision on provisioning the resources (IaaS), and finding the adequate configurations for their workloads (PaaS), probably without any dynamic scaling out and with higher costs.

The remainder of this paper is structured as follows. In the next section, we present our scheduling planner. The design and implementation of our framework follow in Section 3, and its experimental evaluation goes in Section 4. Related work is discussed in Section 5, and the paper concludes in Section 6.

2. SCHEDULING PLANNER

Scheduling, whether it is located at the IaaS or PaaS level, is a core activity in cloud computing that impacts the overall system performance and utilization. Due to the inherent dependencies between computation and data, scheduling workflow tasks is generally more difficult than scheduling embarrassingly parallel jobs. As stated before, most Cloud scheduling approaches for workflows aim at single-shot workflow executions and only take into account simple constraints on time and costs. The model we propose, which targets data-intensive workflows for continuous and incremental processing, also enforces constraints over the data communicated between tasks, while still fitting the utility paradigm. Our model implies that data must be shared via NoSQL database, which achieves better performance, scalability, and availability [6]. We first describe our QoD model, and then the scheduling planner which coordinates it.

2.1. Workflow model with QoD

Workflow tasks, with typical WMSs, usually communicate data via intermediate files that are sent from a node to another, or using a distributed file system. Sharing data through a NoSQL database, like in this work, allows us to place data close to the computation nodes more easily than in a relational database.

Our workflow model [7] is differentiated from the other typical models by the following: the end of execution of a task A does not immediately trigger its successor tasks; instead, they should only be triggered when A has generated output with sufficient impact in relation to the terminal task (outcome) of the workflow (which can cause a node being executed multiple times with the successor nodes being triggered only once). For example, a workflow that is constantly processing data coming from a network of temperature sensors, to detect fires in forests, would not need to be always computing tasks (e.g. calculating hotspots, updating the risk level) whose output would not change significantly in the presence of small jitters in temperature. The workflow will only issue a displacement order to a fire department if more than a certain number of sensors have detected a steep increase in temperature. This way, tasks would only need to specify the minimum impact that their input data needs to have that is worth their execution toward final outcomes.

The level of data changes necessary to trigger a task, denoted by QoD bound κ, is specified through multi-dimensional vectors that associate QoD constraints with data containers, such as a column or group of columns in a table of a given column-oriented database. κ bounds the maximum level of changes through numeric scalar vectors defined for each of the following orthogonal dimensions: time (θ), sequence (σ) and value (ν).

Time Specifies the maximum time a task can be on hold (without being triggered) since its last execution occurred. Considering $\theta(o)$ provides the time (e.g. seconds) passed since the last execution of a task, that is dependent on the availability of data in the object container o, this time constraint κ_θ enforces that $\theta(o) < \kappa_\theta$ at any given time.

Sequence Specifies the maximum number of updates that can be applied to an object container o without triggering a task that depends on o. Considering $\sigma(o)$ indicates the number of applied updates over o, this sequence constraint κ_σ enforces that $\sigma(o) < \kappa_\sigma$ at any given time.

Value Specifies the maximum relative divergence between the updated state of an object container o and its initial state, or against a constant (e.g. top value), since the last execution of a task dependent on o. Considering $\nu(o)$ provides that difference (e.g. in percentage), this value constraint κ_ν enforces that $\nu(o) < \kappa_\nu$ at any given time. It captures the impact or importance of updates in the last state.

A QoD bound can be regarded as an SLA, defining the minimum performance required for a workflow application that is agreed between consumers and providers.

2.2. Abstract scheduling planner

Generally, scheduling workflow tasks is a NP-complete problem. Therefore, we provide here an approximation heuristic that attempts to minimize the costs based on local optimal solutions. The QoD bounds are involved in this process to offer price flexibility, which is very important for continuous processing.

We state the problem as a coordinator node attempting to map a workflow graph G to available worker nodes in a way that minimizes costs and yet respects time and QoD constraints. A single execution of each workflow graph must be completed until a specified time limit L (e.g. in minutes). A task T has a specification in terms of its complexity and tolerated relaxation QoD. This complexity represents the computational cost a task has for being executed in relation to a standard task in a standard machine (this section abstracts from such details, they are given in Section 3). Tolerated relaxation consists of the QoD constraints that are associated with the input data fed to each task.

Worker machines have a specification in terms of their current capability and reference price. This capability is the power of the machine with its current load availability (capability calculation is given in Section 3). Reference price is a standard value that is then adjusted for current availability and load usage of each worker.

The scheduling planning can be divided into two phases. First, tasks are organized into branches (e.g. Fig. 1): connected tasks where each has exactly one predecessor and one successor, except from the last task which can have multiple successors (i.e. pipeline). Branches are ordered by their summed complexity. Tasks that do not fit in the pipeline, are still treated as a pipeline, albeit with a single task within. This means that such tasks will be simply allocated to workers offering the best cost for them.

Secondly, inner branch scheduling is performed by starting from the most complex branch to the least complex one. To schedule tasks inside a branch in an optimal manner, we decompose the problem into a Markov Decision Process (MDP) [8], since it is a common and proven effective technique for sequential decision problems (e.g. [9]).

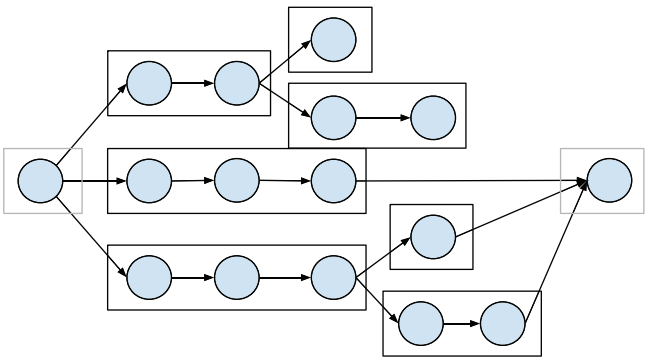

FIGURE 1. Branches in a workflow DAG.

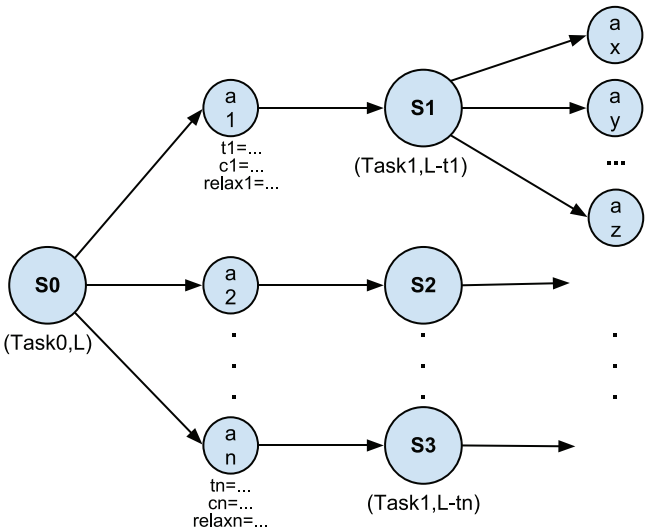

FIGURE 2. Markov decision process diagram.

Briefly, a MDP consists of a set of possible states S, a set of possible actions A, a reward function $R(S, a)$ and a transition model $Tr(S, a, S')$ describing each action's effects in each state. Since R values are guaranteed in our problem, we use deterministic actions instead of stochastic actions, i.e. for each state and action we specify a new state ($Tr : S \times A \rightarrow S'$). The core problem of MDP is to find an optimal policy $\pi(S)$ that specifies which action to take for every state S.

Figure 2, depicts a diagram representing the decomposition of the problem. Each state S in the model corresponds to a task and a time limit to the workflow makespan. Actions represent the allocation of tasks to VM slots in workers. When an action is taken, an immediate reward is given, i.e. three variables specifying the time taken for one execution, the reference cost per hour, and the minimum relaxation of data freshness, within specified QoD limits, that assures the lowest price.

Finding the optimal policy π for each state S (i.e. choosing the right action a to take when on state S) consists of minimizing the cumulative cost of the rewards obtained when

transitioning from S to a terminal state. Hence, we only know the reward $R(S, a)$ after following all possible transitions from state S', such that $S \times a \rightarrow S'$, to a final state. Nonetheless, the processing time, retrieved from the immediate reward of an action a, is discounted from the time limit L when transitioning from S to S' through a. If L is zero or lower in a state S, all paths going through S are cut and it is necessary to find other paths. If there is no other path, it means that it is not possible to compute all tasks in the specified time limit. This minimization problem can be described by Equation (1), where n is the number of tasks, m is the total number of available VM slots, tc is the task complexity, c is the worker cost for unitary task, r the tolerated relaxation due to the QoD enforcement, wc is the worker capacity and x is a function that returns 1 if slot j is available and 0 otherwise.

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} \underbrace{tc_i c_j (1-r)}_{\text{cost}} x_j \\ \text{subject to} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} \underbrace{\frac{tc_i}{wc_j}}_{\text{time}} x_j <= L, \quad x_j \in \{0,1\} \end{aligned} \tag{1}$$

To solve this optimization problem and optimally allocate tasks to workers (i.e. with overall lowest cost and yet respecting time and QoD constraints), we developed a dynamic programming algorithm as described in Listing 1. We elaborate in the following:

> *Lines 2–3* contain the stop condition, when there are no more tasks/states to follow;
> *Lines 4–10* check, through the use of a cache, whether a certain path was already explored before;
> *Lines 11–20* contain the transition of states, thereby exploring all actions of a current state (which is represented by *task* and *totalTime*);
> *Lines 16–17, 21–22* check for whether the time limit was violated or not, causing the algorithm to explore other actions at the same level;
> *Lines 23–27* store the minimum cost found for the current state;
> *Lines 28–29* correspond to optimization code that caches already explored paths and respective minimum costs and times. Additionally, when a slot is locked (line 18) it can no longer be used by successor tasks.

This algorithm runs in $\mathcal{O}(w^t)$, where w is the number of workers and t the number of tasks. Some optimizations were performed, namely caching the rewards of states, obtained by traversing the sub-graphs until the terminal state in the MDP model. The whole process of planning and scheduling is synthesized in the following:

1. Discover available workers and request cost and expected completion time for every task. These values should be guaranteed for a certain time frame, which should be longer than the time taken to perform the planning and allocate tasks.
2. Divide the workflow in pipelines.
3. Divide the overall time limit L per each pipeline and weighted by their summed complexity.
4. Generate scheduling plans for each pipeline, starting from the most complex and ending with the least complex.
5. Allocate tasks to workers according to the generated plans.
6. Start workflow execution and repeat steps 1, 4 and 5 if any worker fails or periodically according to user configurations.

2.3. Prototypical scenario

As a motivational example we describe a data processing workflow, for continuous and incremental processing, that expresses a simulation of a prototypical scenario inspired by the calculation of the Air Quality Health Index (AQHI),[2] used in Canada. It captures the potential human health risk from air pollution in a certain geographic area, typically a city, while allowing for more localized information. The incoming data fed to this workflow is obtained through several detectors comprising three sensors to gauge the amount of Ozone (O_3), Particulate Matter ($PM_{2.5}$) and Nitrogen Dioxide (NO_2).

Figure 3 illustrates the workflow with the associated QoD vectors and the main columns (some columns were omitted for readability purposes) that comprise the data containers in which the processing steps' triggering depends on. k specifies:

1. the maximum time, in hours, the step can be on hold;
2. the minimum amount, in percentage, of changes necessary to the triggering (e.g. 20% associated to step C means that this will be triggered when at least 20% of the detectors have been changed by step B);
3. the maximum accepted divergence, in units.

We describe each processing step in the following:

Step A. This step continuously feeds data to the workflow by reading sensors from detectors that perceive changes in the atmosphere to simulate asynchronous and deferred arrival of update sensory data. The values from each sensor are written in three columns (each row is a different detector) which are grouped as a single data container with one associated k.

Step B. Calculates the combined concentration (of pollution) of the three sensors for each detector whose values were changed in the previous step. Every single calculated value is written on column *concentration*.

Step C. Processes the concentrations of small areas, called zones, encircled by the previously changed detectors. These

[2] www.ec.gc.ca/cas-aqhi/

```
def min_cost(tasks, workers, totalTime, timeLimit):
  if not tasks:
    return 0, 0, []
  t = tasks[0]
  str0 = ''
  for ww in workers:
    str0 += str(ww.slots)
  key = t.name + str0
  if key in cache:
    return cache[key]
  minCost, minCostTime, minCostPath = float('inf'), None, []
  for w in workers:
    if not w.slots:
      continue
    time = calculate_time(t, w)
    if(totalTime + time > timeLimit):
      continue
    w.slots -= 1
    v1, v2, v3  = min_cost(tasks[1:], workers, totalTime + time, timeLimit)
    w.slots += 1
    if v2 == None | totalTime + time + v2 > timeLimit:
      continue
    totalCost = calculate_cost(t, w) + v1
    if totalCost < minCost:
      minCost = totalCost
      minCostTime = time + v2
      minCostPath = [w.name] + v3
    if minCostPath:
      cache[key] = minCost, minCostTime, minCostPath
  return minCost, minCostTime, minCostPath
```

Listing 1. Scheduling Algorithm

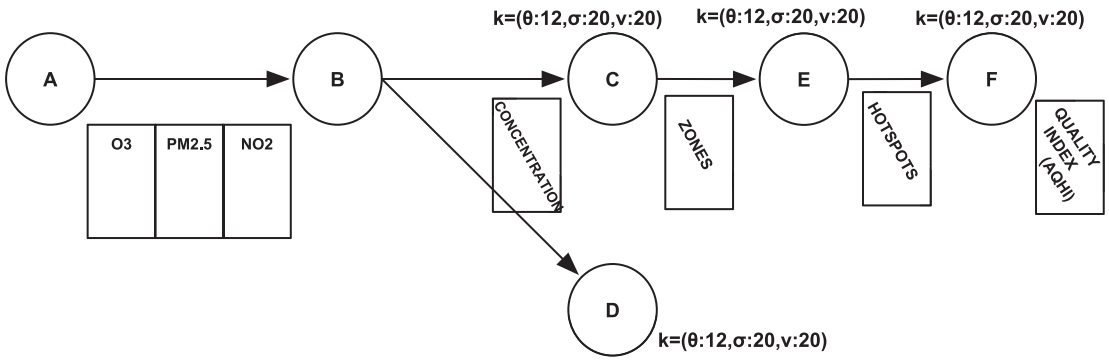

FIGURE 3. Workflow example.

zones can be regarded as small squares within the overall considered area and comprise the adjacent detectors (until a distance of two in every direction). The concentration of a zone is given by a simple multiplicative model of the concentration of each comprising detector.

Step D. Calculates the concentration of points of the city between detectors, thereby averaging the concentration perceived by surrounding detectors; and plots a chart containing a representation of the concentrations throughout the whole probed area, for displaying purposes, and reference

of concentration and air quality risk indicator in localized areas of a city.

Step E. Analyses the previous stored zones and respective concentrations in order to detect hotspots; i.e. zones where the overall concentration is above a certain reference. Zones deemed as hotspots are stored in column *hotspots* for further analysis.

Step F. Performs final reasoning about the hotspots detected, thereby combining, through a simple additive model, the amount (in percentage) of hotspots identified with the average concentration of pollution on all hotspots. Then, the AQHI index is produced and stored for each wave of incoming data.

3. WAAS DESIGN AND IMPLEMENTATION

In this section, we describe our proposed prototypical middleware framework that embodies the vision of a WMS at the PaaS level, that we call Workflow-as-a-Service (or WaaS). We approach its main design choices and the more relevant implementation details. We address: (i) workflow description and WMS integration, (ii) the cost model, and (iii) how resource allocation is enforced.

Figure 4 depicts the WaaS distributed network architecture in the Cloud, where workflows are set up to be executed upon a cluster of worker machines connected through a local, typically high-speed, network. A designated coordinator machine, running the WaaS server VM instance, is in charge of allocating workflow tasks to available worker nodes (according to a scheduling algorithm), and collect monitoring information regarding node load and capacity.

The input/output data are shared among tasks via a shared columnar noSQL data store. Each worker node executes the workflow tasks scheduled to it as guest VM instances, using Xen or QEMU/KVM [10] images, and in particular, a Xen (or QEMU/KBM) virtual appliance with Linux OS, a JVM and a QoD-enabled middleware for cloud noSQL storage.

The WaaS middleware carries out three major steps in its operation. First, according to the workflow descriptions, WaaS performs the planning by exploring the scheduling alternatives for the workflow tasks and branches, carrying out the algorithm described in Section 2. Then, according to the schedule calculated, it performs the allocation of resources at nodes, by assigning the corresponding VMs for tasks at nodes, according to their cost and available capacity. The workflow is then started, and tasks continually re-executed according the QoD parameters defined as new input becomes available and considered.

Additionally, all nodes inform the coordinator only of relevant changes in their available capacity, so that the coordinator can adjust and fine-tune scheduling and allocation decisions, since the coordinator makes use of declarative information stating resource requirements for tasks. When new

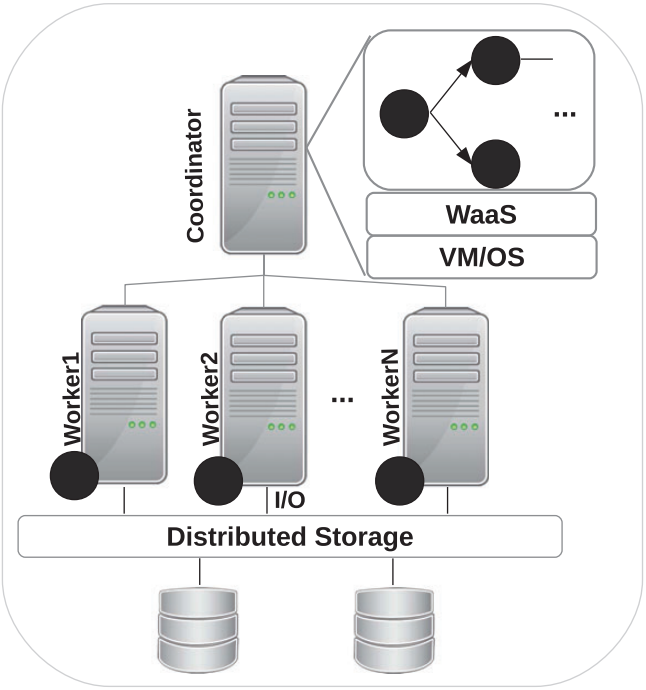

FIGURE 4. WaaS network architecture.

nodes are added to the cluster or become unavailable, the scheduling must also be recalculated.

Regarding the state of the QoD control variables, it is stored in memory for efficiency. If a worker machine goes down, another node taking its place should execute as soon as possible (without postponing) to reset the QoD state. This way, we ensure QoD constraints are never violated, albeit there is a small performance penalty (which is of lesser importance when compared to the overhead of persisting state in the database or filesystem).

3.1. Workflow description and WMS integration

Workflow specification files need to be enhanced to include declarative information required for the scheduling. This is currently defined with special keywords in the workflow descriptions of DAGMan [11] files, that are parsed by the WaaS framework. They must contain the description of the workflow graph where each processing step (to be executed as a task) is annotated specifying explicitly the underlying data containers in the noSQL storage (e.g. tables, columns, rows by ID or predicate or combinations of any of these) it depends on for its input. More precisely, we introduced the following new keyword in a DAG input file: QOD *JobName DataContainer Time Sequence Value* where: *JobName* is the name of the processing step that we want to make asynchronous; *DataContainer* is the table, column or row in the underlying data store on which the previously

described processing step depends; *Time* (seconds), *Sequence* and *Value* (percentage, e.g. 0.2 or 20%) are the QoD constraints defined in Section 2.1. A simple usage example is shown in Listing 2. Note that job D is only triggered when both data containers *column1* and *column2* comply with the specified QoD constraints. This approach is used throughout as it preserves transparency and compatibility when workflows are deployed in other, non-enhanced WMS.

```
JOB   A   A.condor
JOB   B   B.condor
JOB   C   C.condor
JOB   D   D.condor
PARENT A CHILD B C
PARENT B C CHILD D
QOD B column0 3600 30 0.15
QOD C column0 3600 30 0.15
QOD D column1 3600 20 0.1
QOD D column2 3600 20 0.1
```

Listing 2. DAG Description

Regarding failure handling and cluster membership, if a node fails or any time a node enters or parts, the scheduling is recalculated. Like with any other mainstream WMS, exceptions and unexpected task termination, whether gracefully or not, can be easily detected by the task container (e.g. YARN[3]). Thus, we are able to quickly detect the interrupted task and reschedule it on available resources. This is carried out with two alternatives: either simply following a best-effort approach (i.e. only rescheduling failing tasks to other available worker machines), or by making a new scheduling plan (as described in Section 3.1).

As for byzantine behavior, incorrect processing or corruption of data, this is obviously a very relevant and difficult problem (not addressed by popular WMSs), but it is outside of the scope of this particular work. A possible avenue to explore in the future could be to incorporate transactional support in the cloud storage, as well as redundant executions with quorum verification.

With respect to tasks that do not make any progress or get stuck, the only mechanism that we have against that, also followed by other WMSs, is having user-defined timeouts for each different task. In any case, note that data loss is unlikely, since we separate the application logic from the database and we assume storage replication is enabled, as it is default in HBase.

3.2. Cost model

The cost model of WaaS is based on considering task complexity and dynamic price definition. Assessing task complexity regarding processing and memory requirements has been explored in previous works [12–14]. Regarding CPU and memory requirements, the base approach is inspired in CloudSim [15] and uses declarative definitions of MIs (millions of instructions) and MBs of memory required. Additionally, we leverage previous executions of tasks in a machine (e.g. one of the nodes) against the requirements from a reference workload, a unitary cost task (that serves as reference of a task with unitary cost); e.g. Linpack benchmark (as used in [12]), that can also be used to rank the relative capacity of different worker nodes among them and against a reference one.

Regardless of the approach employed, we can determine an estimate on how long each task will take to complete with a given capacity awarded in the node (i.e. time = task complexity/worker capacity). More than one task may share a node's resources for execution, but while ensuring resource and performance isolation, as described in the next subsection.

In the general case where the infrastructure is shared by many users and workflows, the price of executing each task is calculated depending on the resources required pondered with the overall system load.

There is price elasticity: when resources are scarce or there are many users, unitary prices increase, otherwise, when resources are overabundant, prices decrease, with a reference price, as previously addressed in P2P grids [16].

Usually, the cost of executing a workflow for the first time, will be the sum of the cost of executing its tasks. In the continuous execution model of WaaS, although input is being updated or new input being provided (e.g. sensory data), tasks are only re-executed when QoD parameters are reached. Therefore, the saved executions (i.e. task executions that are avoided until QoD is reached) will imply a lower total cost for a given number of workflow executions.

Additionally, since the interval between consecutive executions of a given task can be significant, there is no point in paying (regardless of it being real money or some form of credits) according to the common cloud cost model of VM hours of execution, as these may be idle the majority of time. Therefore, we implement a service where task executions are incurred only for the time of execution, plus a *tax* of 10% to account for the overhead of reusing resources by switching among guest VM instances that execute different tasks, possibly from different workflows.

3.3. Resource allocation and isolation

As already said, resources at nodes are engaged as virtual machine (VM) instances, in particular with images derived from virtual appliances described above. Thus, when the scheduler decides to allocate a VM based on a task's requirements and price constrains, it essentially aims at two things: (i) allocate enough resources for the task and (ii) ensure

[3] http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

that those resources and their availability are not hindered by the scheduling of other tasks in the same node. Note that no more than one task is permitted to be executed inside the same VM. We make extensive use of virtualization technology to allow such fine-grained allocation and acceptable performance isolation guarantees.

The VM instances can be preconfigured and prelaunched, ready to execute a given workload, and by means of the WaaS component installed, can later execute the workload of another task, without the need of being shutdown and rebooted, easing resource sharing and reducing the amount of wasted resources. Therefore, we configure the hypervisor in Xen to cap the percentage of physical CPU(s) and physical memory awarded to a given VM according to the scheduling decided. This can be repeated until the node capacity is fully allocated, with a 10% safety quota for middleware own operation. This can also be achieved, albeit with less flexibility by parameterizing QEMU/KVM. This ensures that when a task is scheduled to a node, the resources it is expected to make use of, are not in contention with the resources required by other tasks executing at the same time. Any degradation will be graceful and only when contention is very high.

Recall that worker top capacity is established assessing the performance of a reference workload against the performance of the same workload against a reference machine. Regarding instantaneous available capacity at a node, in order to fine-tune the information driving the scheduling (that is aware of VM allocations at each node) we resort to the SIGAR[4] library that has enough precision and is actually platform-independent.

4. EVALUATION

This section presents experimental evaluation that was carried out to show the benefits of our approach, in simulated environment as well in a realistic scenario using the prototypical workflow described in Section 2.3. In particular, to assess whether our model can effectively reduce costs, while complying with deadlines, and use relaxation (corresponding to the percentage of saved executions with the enforcement of QoD constraints).

All tests were conducted using six machines with an Intel Core i7-2600K CPU at 3.40 GHz, 11926 MB of available RAM memory, and HDD 7200RPM SATA 6 Gb/s 32 MB cache, connected by 1 Gigabit LAN.

We compared three different approaches with our algorithm: Greedy-time, Greedy-cost and Random. Greedy-time selects for each task the worker that offers the minimum processing time at that moment. Similarly, Greedy-cost selects at each step the worker that offers the minimum processing cost. And Random selects a random worker for each task.

[4] http://support.hyperic.com/display/SIGAR/Home

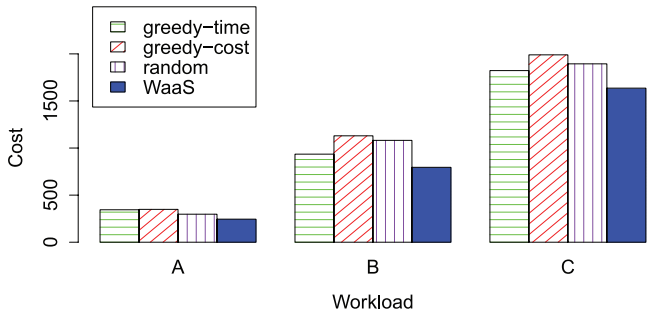

FIGURE 5. Cost per hour taken for pipeline execution.

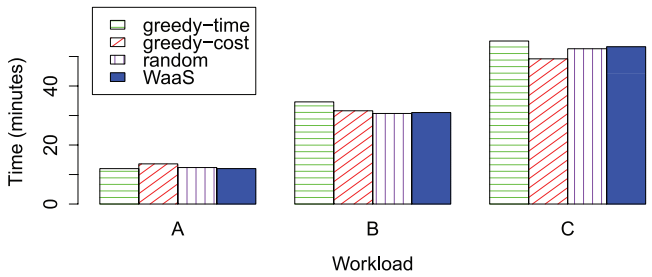

FIGURE 6. Time taken for a single pipeline execution.

We conducted a simulation, built in Python, to compare our model with different approaches. Note that this simulation corresponds to the isolation of the coordinator machine, so that it can be properly evaluated without the interference (delays) of worker machines (i.e., tasks complexity and workers capacity are synthetic). We generated hundreds of pipelines with 5, 10 and 15 tasks, corresponding to workloads A, B and C, respectively. Note that the payload of the intrinsic tasks was dummy content (i.e., we were only interested in the task metadata for the coordinator scheduling). Inside each workload, results were averaged to reduce noise.

Figure 5 shows that our model, WaaS, can effectively reduce costs. The gains are higher when there is more variance in the worker's cost. The costs achieved by our model, represent the critical path of the MDP model, and, since no time limit was imposed, they are undoubtedly the minimum possible costs for the considered workloads.

Figure 6 shows that the time obtained with WaaS for a single pipeline execution is not much different from the remaining approaches. Lower costs often mean that workers with lower capabilities were used, and therefore the makespan was higher.

Figure 7 illustrates the correlation observed between time (makespan) and cost for 1000 samples of different pipelines with 10 tasks and in diverse worker settings. Each sample, consisting of a different set of tasks and workers, was executed for the four different algorithms, and we can observe that the cost increases with the time. Unsurprisingly, this happens due to the cost and time functions being directly proportional with

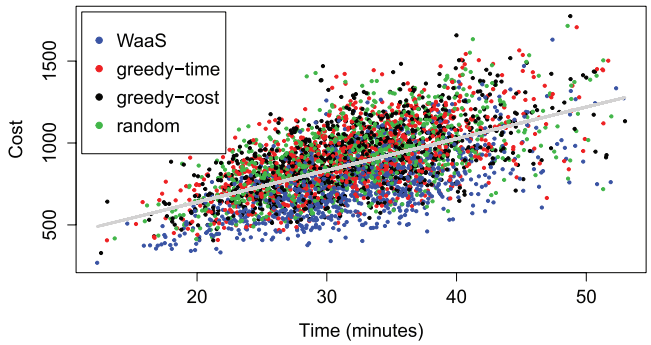

FIGURE 7. Time cost correlation for 1000 samples.

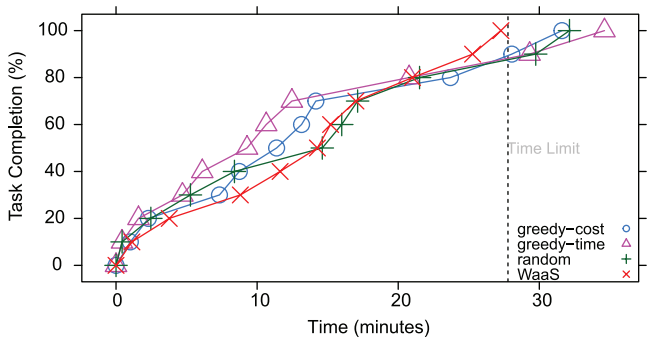

FIGURE 8. Task completion over time.

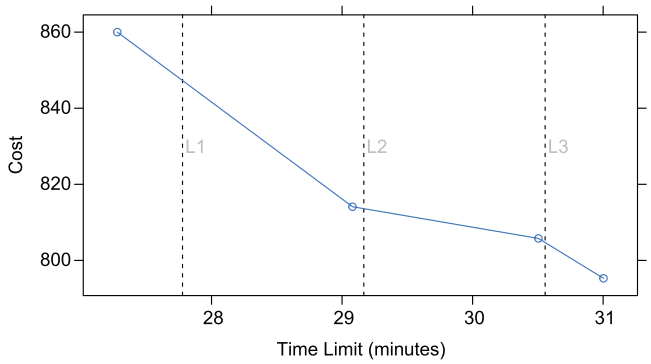

FIGURE 9. Cost variation for different time limits.

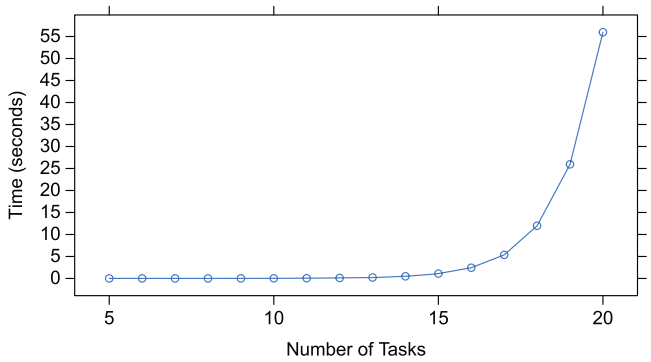

FIGURE 10. Time taken for planning.

the task complexity. WaaS appears always at the bottom (blue points) with the lower costs, as expected.

Through Fig. 8, we may observe that our algorithm with WaaS exhibits the highest task completion rate within the time limit, while others fail to process the complete workflow inside specified time frames (i.e., roughly the last 20% of tasks are processed outside of the deadline). However, there is a price to pay when such time frames are shrunk, as shown in the next figure.

Figure 9 depicts how costs vary with the imposed time limits L_1, L_2 and L_3. We can see that costs decrease with the expansion of time limits. There is a point beyond which extending more the deadline does not reduce the costs, which corresponds to the time taken to go through the critical path, the one that provides the lowest cost, in the MDP graph. Also, when the time limit is lower than the MDP path with the minimum time, it is not possible to complete the whole pipeline tasks inside the limit. Thus, there is an interval of time within which users can adjust the limits.

Figure 10 shows the time evolution for planning with pipelines with different number of tasks and workers (for simplicity, the number of workers is the same as the number of tasks). Although we performed optimizations with the MDP-based algorithm, we may see that time follows an exponential tendency with the number of tasks, like stated in Section 2.2. For less than 15 pipeline tasks the times obtained are negligible, and for more than 17 tasks the times start to increase drastically (above 10 s). However, workflows containing more than 10 tasks in pipeline are not common.[5] Furthermore, there is still space for optimization and parallelization on our MDP-based algorithm.

We can see in Fig. 11 how the cost varies with the level of relaxation for a pipeline with 10 tasks where each was set to have levels of relaxation of 0 (no relaxation), 15, 30 and 45%. The cost decreased down to 233 units with 45% of relaxation.

Using our simple WMS, we ran the prototypical workflow presented in Section 2.3 where tasks consisted of Java applications that read and modify data in the NoSQL database. Next, we show how relaxation, which corresponds to avoided executions and saved resources, is obtained from a workflow model with QoD enforcement.

Figure 12 shows that the number of executions decreases in an almost linear way as the allowed percentage of changed detectors (σ) increases. When σ was 25% we saved 20% of 168 executions (i.e., fewer 33 executions than using regular DAG semantics); and for 80% of detected changes we only performed 80 executions (48%). The machine loads and

[5] https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

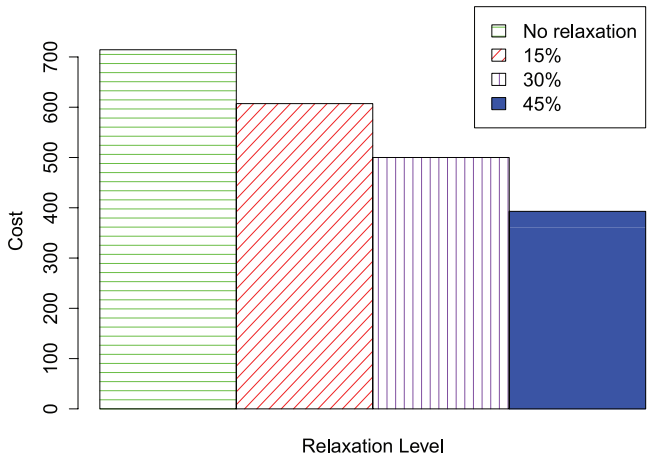

FIGURE 11. Cost versus relaxation.

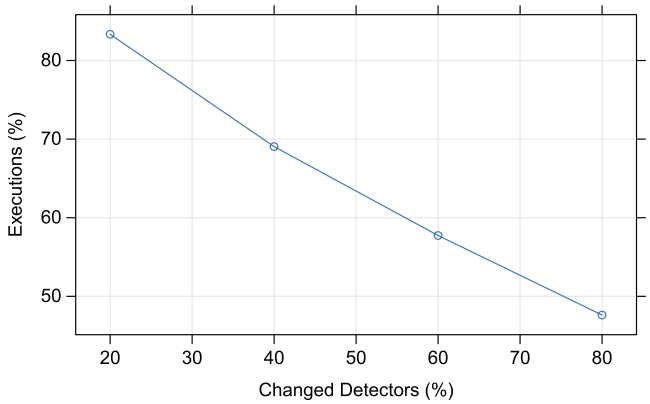

FIGURE 12. Saved executions in processing step that calculates zones.

resource utilization were naturally proportional to the savings presented here.

We can observe the resulting impact in the percentage of number of executions, when combining the QoD of steps C and E (i.e., minimum percentage of changes in zones and detectors), as illustrated in Fig. 13. For the particular trial of step E (which calculates and identifies hotspots): (i) presents an improvement, almost linear, in the number of executions when no QoD is enforced on step C; and (ii) only improves starting from 75% when QoD is enforced for the detectors. In a workflow with pipeline processing, like the one considered, it is natural that the QoD of previous or upstream steps influence the executions of current and downstream steps in the pipeline, since the inputted data are derived from upstream, i.e., from the beginning of the processing.

Later, we predetermined the value of the QoD for the former steps in the workflow, and studied the gains obtained, regarding the number of times that step F is executed (Fig. 14). A great amount of executions were saved, even for 20% of changed

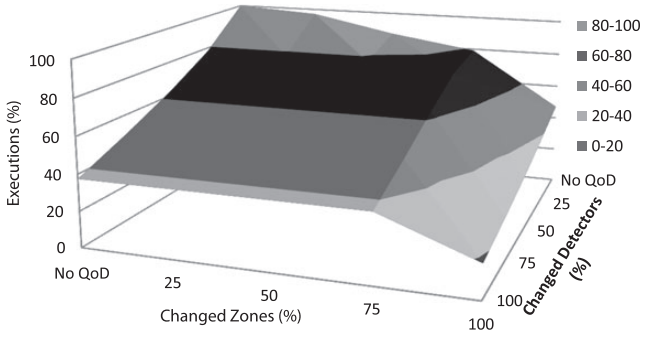

FIGURE 13. Saved executions.

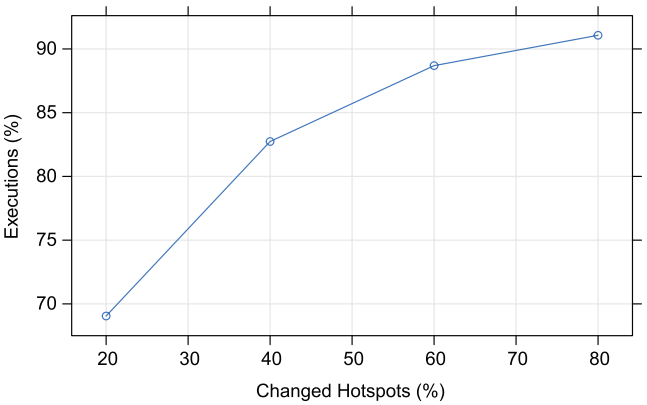

FIGURE 14. Saved executions in processing step that calculates AQHI.

hotspots where 70% of the total executions without any QoD (i.e., 168 executions) were spared. At 80% of changed hotspots, only 5% of the total executions were performed with an error not >0.3 (in relation to the regular DAG model). It is natural that, as we go through the actions of the pipeline, the number of executions with QoD is reduced, since the noise from the raw data injected in the workflow is funneled through the processing chain into more refined and structured data.

5. RELATED WORK

Much work has been done regarding scheduling of tasks in grid and cloud settings. A subset of this work targets the scheduling of workflows in particular. For example, [17–19] for grid computing. Our model inherits from and extends the traditional workflow model [20]. Next, we describe some solutions that are closer and more related with our QoD model.

In [21], the authors propose a cost-based workflow scheduling algorithm that is capable of minimizing costs and meeting deadlines for result delivery. Scheduling plans are generated so that tasks are mapped to resource providers according to the services requested (thus following a service-oriented paradigm). It can perform rescheduling if planning

conditions are violated. The tasks are grouped into branches, forming pipelines, but they assume synchronization tasks. These branches are then also processed by a MDP, but over different variables. The impact of data in the results and workflow execution relaxation is not taken into account, unlike in our model. Nonetheless, it has been a common approach to impose time limits and deadline constraints, instead of minimizing execution times [22].

In [23], authors claim that proposed heuristics for scheduling on heterogeneous systems fail by not considering processors with different capabilities. To amend this, authors propose a list-based scheduling algorithm with two distinctive features: the Percentage of Capable Processors effect is taken into account when assigning task node weights; and the adjustment of the effective Earliest Finish Time strategy, by incorporating the average communication cost between the current scheduling node and its children, during the processor selection phase. Our model also takes into account processors with different capabilities for scheduling, since the times and relaxation are calculated based on that within the WaaS environment; however, data impact is also not taken into account on their solution. Also, in [19] authors presented a novel binding scheme to deal with heterogeneity presented in grid and cloud environments, and improve performance by attending to such different characteristics.

In [24], authors have proposed a unified solution that supports multiple scheduling algorithms based on best-effort and advance-reservation approaches, supported by lease scheduling principles and price-based policies. They offer an open source implementation of proposed algorithms, Haizea, but it lacks on efficiency, since their effort was more theoretical. Some proposals were made to overcome some of Heizea's limitations like introducing new resource leases and market theory concepts in scheduling decisions [25, 26]. In our approach, we offer flexibility in negotiations by using QoD levels that allow price relaxation.

In [27], an algorithm based on the meta-heuristic optimization technique, particle swarm optimization (PSO), is proposed to minimize the workflow execution cost while meeting deadline constraints. Like WaaS, this work is tailored for cloud environments, addressing issues like heterogeneity and elasticity of the resources. However, unlike WaaS, it does not perform any data reasoning with the aim of saving resources and lowering costs. Besides PSO, other meta-heuristic methods have been proposed, such as genetic algorithms [28] and ant colony optimization [18], which offer satisfactory performance; however, the QoS constraints those algorithms rely on disregard resource efficiency for continuous workflow processing.

In [29], different task scheduling strategies for workflow-based applications are explored. Authors claim that many existing systems for the grid use matchmaking strategies that do not consider overall efficiency for the set of (dependent) tasks to be run. They compare typical task-based greedy algorithms with workflow-based algorithms, that search through the entire workflow. Results show that workflow-based approaches have a potential to work better on data-intensive scenarios, even when task estimates are inaccurate. This comes to strengthen our work, as most scheduling done, which is task-based, does not work well for workflows due to the intrinsic dependencies.

In [30], authors claim that most auto-scaling scheduling mechanisms only consider simple resource utilization indicators and do not consider both user performance requirements and budgets constraints. They present an approach where the basic computing elements are VMs of various sizes/costs, and, by dynamically allocating/deallocating VMs and scheduling tasks on the most cost-efficient instances, they are able to reduce costs. Task-to-VM optimization was also tasked in [31], where a hierarchical scheduling strategy was proposed. Furthermore, advantages of running in a virtual environment, even remotely, over local environments are highlighted here [32]. We also provide a resource utilization metric representing not only the capacity of a worker machine, but also its current load and usage. In addition to this mechanism, we also combined data relaxation which results in good cost savings.

Some solutions attempt to save computing resources, thereby introducing scheduling constraints on energy consumption [33], implementing efficient load balancing techniques [34] or focusing on workload/VM consolidation [35]. While such solutions are complementary to ours, the resource savings are far from the ones achieved with WaaS, since we are capable of avoiding unnecessary task executions.

Other algorithms take into account data, but only for the cost of transmission (e.g. PSO [36]). Since we use a Cloud distributed database where all nodes are interconnected through the same LAN, we do not need to send data directly from a node to another, and therefore this issue is immaterial here.

Furthermore, an example of a currently available related service is the amazon SWF[6] that is growing in popularity. It could also be a target of our work, since it currently includes no reasoning about the impact of computation in data (input and output), as WaaS does, in order to optimize resource usage. However, it deals exclusively with the deployment and scheduling of the tasks in virtualized resources to be run in the cloud.

6. CONCLUSION

This paper makes use of a novel workflow model for continuous data-intensive computing proposing a new Cloud scheduling planner, capable of relaxing prices and respecting time constraints. Our platform gains a special importance in e-science where long-lasting workflows are executed many times often without any new significant or meaningful results (many times only getting noise), wasting monetary funds.

[6] http://aws.amazon.com/swf/

Evaluation results show that our approach is able to reduce costs while respecting time constraints. This cost reduction is higher for larger QoD constraints (which result in larger relaxation). However, larger QoD values can cause higher result deviations, but optimizing that trade-off is out of the scope of this paper.

To the best of our knowledge, no work in the cloud scheduling literature has ever before tried to reason about the data impact on processing steps that cause significant changes on the final workflow outcome for continuous and autonomic processing. Therefore, we believe we have a compelling advancement over the state-of-the-art.

FUNDING

This work was partially supported by national funds through FCT—Fundação para a Ciência e a Tecnologia, under projects PEst-OE/EEI/LA0021/2013, PTDC/EIA-EIA/113613/2009.

REFERENCES

[1] Richards, M., Ghanem, M., Osmond, M., Guo, Y. and Hassard, J. (2006) Grid-based analysis of air pollution data. *Ecol. Model.*, **194**, 274–286.

[2] Brown, D.A., Brady, P.R., Dietz, A., Cao, J., Johnson, B. and McNabb, J. (2007) A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis. In Taylor, I.J., Deelman, E., Gannon, D.B. and Shields, M. (eds), *Workflows for e-Science*, pp. 39–59. Springer, London.

[3] Li, X., Plale, B., Vijayakumar, N., Ramachandran, R., Graves, S. and Conover, H. (2008) Real-time storm detection and weather forecast activation through data mining and events processing. *Earth Sci. Inf.*, **1**, 49–57.

[4] Deelman, E. *et al.* (2006) Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The Cybershake Example. *Proc. 2nd IEEE Int. Conf. e-Science and Grid Computing*, Washington, DC, USA E-SCIENCE'06, p. 14. IEEE Computer Society.

[5] George, L. (2011) *HBase: The Definitive Guide* (1st edn). O'Reilly Media.

[6] Cattell, R. (2011) Scalable SQL and NoSQL data stores. *SIGMOD Rec.*, **39**, 12–27.

[7] Esteves, S., Silva, J.N. and Veiga, L. (2013) Fluchi: a quality-driven dataflow model for data intensive computing. *J. Internet Serv. Appl.*, **4**, 12.

[8] Puterman, M.L. (1994) *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (1st edn). John Wiley & Sons, Inc., New York, NY, USA.

[9] Yih, Y. and Thesen, A. (1991) *Semi-Markov Decision Models for Real-time Scheduling*. Research Memorandum. School of Industrial Engineering, Purdue University.

[10] Bartholomew, D. (2006) Qemu: a multihost, multitarget emulator. *Linux J.*, **2006**, 3.

[11] Couvares, P., Kosar, T., Roy, A., Weber, J. and Wenger, K. (2007) Workflow Management in Condor. In Taylor, I.J., Deelman, E., Gannon, D.B. and Shields, M. (eds), *Workflows for e-Science*, pp. 357–375. Springer, London.

[12] Veiga, L., Rodrigues, R. and Ferreira, P. (2007) Gigi: An Ocean of Gridlets on a 'Grid-For-The-Masses'. *7th IEEE Int. Sym. Cluster Computing and the Grid, 2007. CCGRID 2007*, pp. 783–788.

[13] Simão, J. and Veiga, L. (2012) Qoe-jvm: An Adaptive and Resource-Aware Java Runtime for Cloud Computing. In Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S. and Cruz, I. (eds), *On the Move to Meaningful Internet Systems: OTM 2012*, Lecture Notes in Computer Science 7566, pp. 566–583. Springer, Berlin, Heidelberg.

[14] Costa, F., Silva, J.N., Veiga, L. and Ferreira, P. (2012) Large-scale volunteer computing over the internet. *J. Internet Serv. Appl.*, **3**, 329–346.

[15] Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C. A.F. and Buyya, R. (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.*, **41**, 23–50.

[16] Oliveira, P., Ferreira, P. and Veiga, L. (2011) Gridlet Economics: Resource Management Models and Policies for Cycle-Sharing Systems. In Riekki, J., Ylianttila, M. and Guo, M. (eds), *Advances in Grid and Pervasive Computing*, Lecture Notes in Computer Science 6646, pp. 72–83. Springer, Berlin, Heidelberg.

[17] Wieczorek, M., Prodan, R. and Fahringer, T. (2005) Scheduling of scientific workflows in the Askalon grid environment. *SIGMOD Rec.*, **34**, 56–62.

[18] Chen, W.-N. and Zhang, J. (2009) An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. *IEEE Trans. Syst. Man Cybern. C: Appl. Rev.*, **39**, 29–43.

[19] Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Mellor-Crummey, J., Liu, B. and Johnsson, L. (2005) Scheduling Strategies for Mapping Application Workflows onto the Grid. *Proc. 14th IEEE Int. Sym. High Performance Distributed Computing, 2005. HPDC-14*, pp. 125–134.

[20] Yu, J. and Buyya, R. (2005) A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, **34**, 44–49.

[21] Yu, J., Buyya, R. and Tham, C.K. (2005) Cost-Based Scheduling of Scientific Workflow Application on Utility Grids. *Proc. 1st Int. Conf. e-Science and Grid Computing*, Washington, DC, USA E-SCIENCE'05, pp. 140–147. IEEE Computer Society.

[22] Eder, J., Panagos, E. and Rabinovich, M. (1999) Time Constraints in Workflow Systems. In Jarke, M. and Oberweis, A. (eds), *Advanced Information Systems Engineering*, Lecture Notes in Computer Science 1626, pp. 286–300. Springer, Berlin, Heidelberg.

[23] Shi, Z. and Dongarra, J.J. (2006) Scheduling workflow applications on processors with different capabilities. *Future Gen. Comput. Syst.*, **22**, 665–675.

[24] Sotomayor Basilio, B. (2010) Provisioning Computational Resources Using Virtual Machines and Leases. Ph.D. Thesis, Chicago, IL, USA, AAI3419776.

[25] Sotomayor, B., Montero, R.S., Llorente, I.M. and Foster, I. (2008) Capacity Leasing in Cloud Systems using the

OpenNebula Engine. *Cloud Computing and Applications 2008 (CCA08)*.

[26] Yang, Z., Yin, C. and Liu, Y. (2011) A Cost-Based Resource Scheduling Paradigm in Cloud Computing. *Proc. 2011 12th Int. Conf. Parallel and Distributed Computing, Applications and Technologies*, Washington, DC, USA, PDCAT'11, pp. 417–422. IEEE Computer Society.

[27] Rodriguez, M. and Buyya, R. (2014) Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds. *IEEE Trans. Cloud Comput.*, **2**, 222–235.

[28] Ye, Z., Zhou, X. and Bouguettaya, A. (2011) Genetic Algorithm Based QoS-Aware Service Compositions in Cloud Computing. In Yu, J., Kim, M. and Unland, R. (eds), *Database Systems for Advanced Applications*, Lecture Notes in Computer Science 6588, pp. 321–334. Springer, Berlin, Heidelberg.

[29] Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A. and Kennedy, K. (2005) Task Scheduling Strategies for Workflow-Based Applications in Grids. *Proc. 5th IEEE Int. Sym. Cluster Computing and the Grid (CCGrid'05)*, Washington, DC, USA, pp. 759–767. IEEE Computer Society.

[30] Mao, M. and Humphrey, M. (2011) Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–12.

[31] Wu, Z., Liu, X., Ni, Z., Yuan, D. and Yang, Y. (2013) A market-oriented hierarchical scheduling strategy in cloud workflow systems. *J. Supercomput.*, **63**, 256–293.

[32] Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B. and Good, J. (2008) On the Use of Cloud Computing for Scientific Workflows. *IEEE 4th Int. Conf. eScience, 2008. eScience'08*, pp. 640–645.

[33] Fard, H., Prodan, R., Barrionuevo, J. and Fahringer, T. (2012) A Multi-Objective Approach for Workflow Scheduling in Heterogeneous Environments. *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May, pp. 300–309.

[34] Li, J., Peng, J. and Zhang, W. (2011) An energy-efficient scheduling approach based on private clouds. *J. Inf. Comput. Sci.*, **8**, 716–724.

[35] Feller, E., Rilling, L. and Morin, C. (2011) Energy-Aware Ant Colony Based Workload Placement in Clouds. *2011 12th IEEE/ACM Int. Conf. Grid Computing (GRID)*, September, pp. 26–33.

[36] Pandey, S., Wu, L., Guru, S. and Buyya, R. (2010) A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments. *2010 24th IEEE Int. Conf. Advanced Information Networking and Applications (AINA)*, pp. 400–407.