Planning and Scheduling Data Processing Workflows in the Cloud with Quality-of-Data Constraints*

Sérgio Esteves and Luís Veiga

Instituto Superior Técnico - ULisboa
INESC-ID Lisboa, Distributed Systems Group, Portugal
sesteves@gsd.inesc-id.pt · luis.veiga@inesc-id.pt

Abstract. Data-intensive and long-lasting applications running in the form of workflows are being increasingly more 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 stark challenge, we introduce a new scheduling criterion, Quality-of-Data (QoD), which describes the requirements about the data that worth 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.

1 Introduction

Data-intensive applications generally comprehend several distinct and interconnected 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 [17], detecting gravitational-waves [3], weather forecasting [12], predicting earthquakes [7], 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.

* 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.

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.

Quality-of-Data (QoD)[1] describes the minimum impact that new input data needs to have in order to trigger 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 causing significant and meaningful changes in the values of final output steps. QoD can also be seen as a metric of triggering relaxation.

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 of virtualization technology and the HBase [10] 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.

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.

[1] *Quality-of-Data* 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.

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. We first describe our QoD model, and then the scheduling planner which coordinates it.

2.1 Workflow Model with Quality-of-Data

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 reduce bandwidth and increase reliability in the presence of failing nodes.

Our workflow model [9] 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 towards 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 (Service-level agreement), 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 in 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.

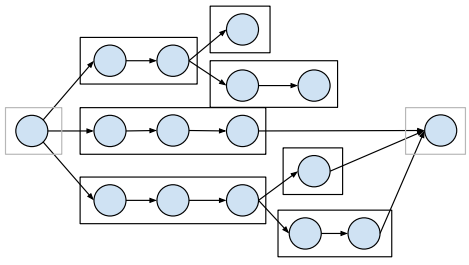

Fig. 1. Branches in a workflow

The scheduling planning can be divided in two phases. First, tasks are organized into branches (e.g., Figure 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.

Second, 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) [16], since it is a common and proven effective technique for sequential decision problems (e.g., [23]).

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.

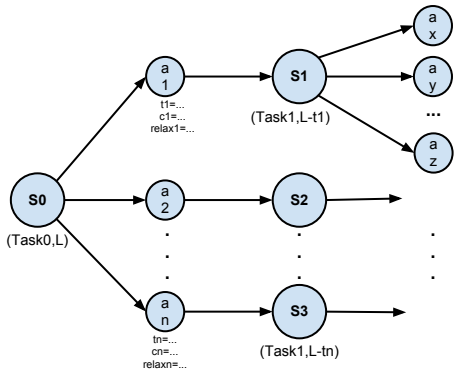

Fig. 2. Markov Decision Process diagram

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., 3 variables specifying the time taken for 1 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.

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, listed as follows.

```
1 def min_cost(tasks, workers, totalTime, timeLimit):
2   if not tasks:
3     return 0, 0, []
4   t = tasks[0]
5   minCost, minCostTime, minCostPath = float('inf'), None, []
6   for w in workers:
7     if not w.slots:
8       continue
9     time = calculate_time(t, w)
10    if(totalTime + time > timeLimit):
11      continue
12    w.slots -= 1
13    v1, v2, v3  = min_cost(tasks[1:], workers, totalTime + time,
          timeLimit)
14    w.slots += 1
15    if v2 == None | totalTime + time + v2 > timeLimit:
16      continue
17    totalCost = calculate_cost(t, w) + v1
18    if totalCost < minCost:
19      minCost = totalCost
20      minCostTime = time + v2
21      minCostPath = [w.name] + v3
22  return minCost, minCostTime, minCostPath
```

Lines 1-5: contain the stop condition, when there are no more tasks/states to follow; lines 6-13: contain the transition of states, thereby exploring all actions of a current state (which is represented by *task* and *totalTime*); lines 10-11, 15-16: check for whether the time limit was violated or not, causing the algorithm to explore other actions at the same level; lines 18-21: store the minimum cost found for the current state. Additionally, when a slot is locked (line 12) 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 roaming the sub-graphs until the terminal state in the MDP model (they were omitted from the algorithm above due to space constraints). 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 higher 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.

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.

We envision a 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 is 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[1] 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 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 nodes are added to the cluster or become unavailable, the scheduling must also be recalculated.

3.1 Workflow Description and WMS Integration

Workflow specification schemas need to be enhanced to include declarative information requiring for the scheduling. This is currently defined with special

comments in the workflow descriptions in DAGMan [6] files, that are parsed by the WaaS framework. They should 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.

This approach is used throughout as it preserves transparency and compatibility where workflows are deployed in other, non-enhanced WMS. Additionally, in particular for the last processing step, it is necessary to specify the desired significance factor: the percentage of variation in the output tabular data that comprises a minimum semantically level of meaning to the workflow users, e.g., 5%. The scheduling is repeated after a predefined parameter of N workflow executions.

Regarding failure handling and cluster membership, if a node fails or every time a node enters or parts, the scheduling is recalculated. Note that all data is saved in the distributed storage (HBase cluster) and WMS can easily restart tasks.

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 [20,19,5]. Regarding CPU and memory requirements, the base approach is inspired in CloudSim 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, e.g., Linpack benchmark (as used in [20]), that can also be used to rank the relative capacity of different worker nodes 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 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 [15].

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 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 instances, in particular with images derived from virtual appliances described above. Thus, when the scheduling decides to allocate a virtual machine based on a task requirements and price constrains, it essentially aims at two things: i) allocate enough resources for the task, and ii) ensure that those resources and their availability are not hindered by the scheduling of other tasks in the same node. 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[2] library that has enough precision and is actually platform-independent.

4 Experimental Evaluation

This sections presents experimental evaluation that was carried out to show the benefits of our approach. In particular, if our model can effectively reduce costs,

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

complying with deadlines, and use relaxation (corresponding to the percentage of saved executions with the enforcement of QoD constraints).

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

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 were dummy content (i.e., we were only interested in the task meta-data for the coordinator scheduling). Inside each workload, results were averaged to reduce noise.

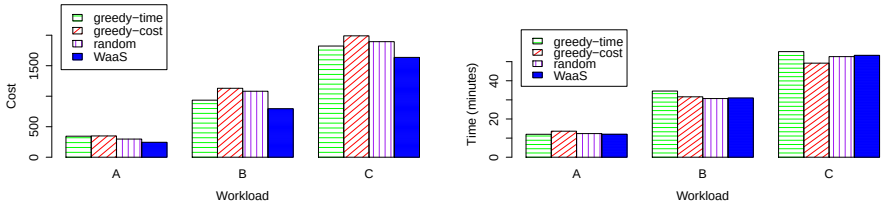

Fig. 3. Cost per hour (left) and time taken for pipeline execution (right)

Figure 3 (left) 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 3 (right) 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 4 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 4 different algorithms, and we can observe that the cost increases with the time. Unsurprisingly, this happens due to the cost and time functions

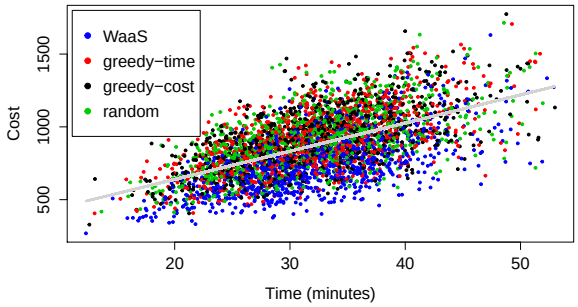

Fig. 4. Time cost correlation for 1000 samples

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

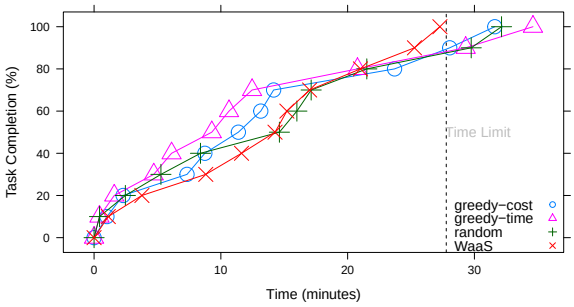

Fig. 5. Task completion over time

Through Figure 5 we may observe that our algorithm with WaaS exhibits the highest task completion rate and is able to meet time limits, 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 6 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 from which expanding 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

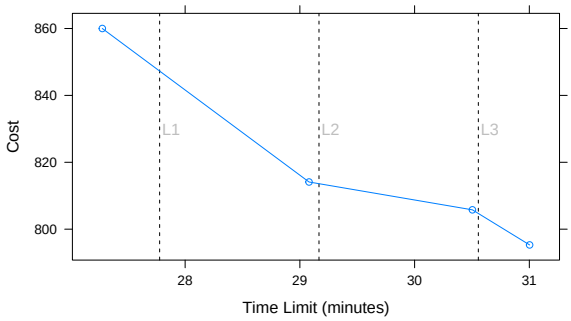

Fig. 6. Cost variation for different time limits

whole pipeline tasks inside the limit. Thus, there is an interval of time within which users can adjust the limits.

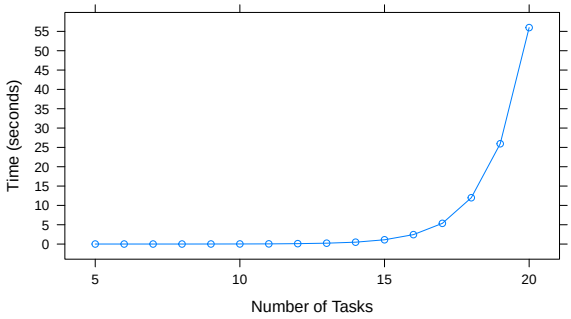

Fig. 7. Time taken for planning

Figure 7 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 seconds). However, workflows containing more than 10 tasks in pipeline are not common.[3] Furthermore, there is still space for optimization and parallelization on our MDP-based algorithm.

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

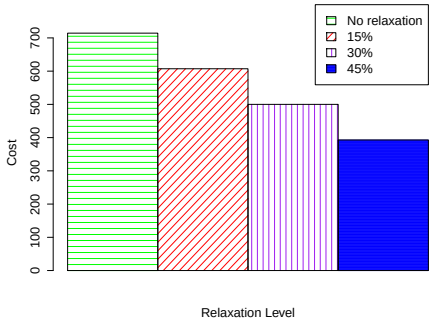

Fig. 8. Cost versus relaxation

We can see in Figure 8 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.

5 Related Work

Many 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, [21,4,13] for Grid computing. Our model inherits from and extends the traditional workflow model [24]. Next, we describe some solutions that are closer and more related with our Quality-of-Data model.

In [25], it is proposed a cost-based workflow scheduling algorithm that is capable of minimizing costs and meeting deadlines for result delivery. A MDP is also used to perform the scheduling, however over different constraints (e.g., tasks can request different services from certain providers). 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, instead of minimizing execution times [8].

In [18], authors claim that proposed heuristics for scheduling on heterogeneous systems fail by not considering processors with different capabilities. 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 in their solution. Also, [13] 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 [2], 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 for 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.

In [14], authors claim that most auto-scaling scheduling mechanism 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 virtual machines (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. This task-to-VM optimization was also tasked in [22], where a hierarchical scheduling strategy was proposed. Furthermore, advantages of running in a virtual environment, even remotely, over local environment are highlighted here [11]. We also provide a resource utilization metric representing not only the capacity of a worker machine, but also its current load usage. In addition to this mechanism we also combined data relaxation which conveys in good cost savings.

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, is proposed. This platform gains a special importance in e-science where long-lasting workflows are executed many times without any new significant and meaningful results (many times only getting noise), wasting monetary funds.

Evaluation results show that our approach is able to reduce costs while respecting time constraints. This cost reduction is higher for larger QoD contraints (which result in larger relaxation). However, larger QoD values can cause higher result deviations, but that problem 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.

References

1. D. Bartholomew. Qemu: a multihost, multitarget emulator. *Linux J.*, 2006(145):3–, May 2006.
2. J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, CCGRID '05, pages 759–767, Washington, DC, USA, 2005. IEEE Computer Society.

3. D. A. Brown, P. R. Brady, A. Dietz, J. Cao, B. Johnson, and J. McNabb. A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis. In I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, editors, *Workflows for e-Science*. Springer London.
4. W.-N. Chen and J. Zhang. An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 39(1):29–43, 2009.
5. F. Costa, J. N. Silva, L. Veiga, and P. Ferreira. Large-scale volunteer computing over the internet. *J. Internet Services and Applications*, 3(3):329–346, 2012.
6. P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger. Workflow management in condor. In I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, editors, *Workflows for e-Science*, pages 357–375. Springer London, 2007.
7. E. Deelman et al. Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '06, pages 14–, Washington, DC, USA, 2006. IEEE Computer Society.
8. J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In M. Jarke and A. Oberweis, editors, *Advanced Information Systems Engineering*, volume 1626 of *Lecture Notes in Computer Science*, pages 286–300. Springer Berlin Heidelberg, 1999.
9. S. Esteves, J. N. Silva, and L. Veiga. Fluchi: a quality-driven dataflow model for data intensive computing. *Journal of Internet Services and Applications*, 4(1):12, 2013.
10. L. George. *HBase: The Definitive Guide*. O'Reilly Media, 1 edition, 2011.
11. C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good. On the use of cloud computing for scientific workflows. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 640–645, 2008.
12. X. Li, B. Plale, N. Vijayakumar, R. Ramachandran, S. Graves, and H. Conover. Real-time storm detection and weather forecast activation through data mining and events processing. *Earth Science Informatics*.
13. A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling strategies for mapping application workflows onto the grid. In *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, pages 125–134, 2005.
14. M. Mao and M. Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–12, 2011.
15. P. Oliveira, P. Ferreira, and L. Veiga. Gridlet economics: Resource management models and policies for cycle-sharing systems. In J. Riekki, M. Ylianttila, and M. Guo, editors, *GPC*, volume 6646 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 2011.
16. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.
17. M. Richards, M. Ghanem, M. Osmond, Y. Guo, and J. Hassard. Grid-based analysis of air pollution data. *Ecological Modelling*, 194(1-3):274 – 286, 2006.
18. Z. Shi and J. J. Dongarra. Scheduling workflow applications on processors with different capabilities. *Future Gener. Comput. Syst.*, 22(6):665–675, May 2006.
19. J. Simão and L. Veiga. Qoe-jvm: An adaptive and resource-aware java runtime for cloud computing. In *OTM Conferences (2)*, pages 566–583, 2012.
20. L. Veiga, R. Rodrigues, and P. Ferreira. Gigi: An ocean of gridlets on a "grid-for-the-masses". In *CCGRID*, pages 783–788. IEEE Computer Society, 2007.

21. M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Rec.*, 34(3):56–62, Sept. 2005.
22. Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63:256–293, 2013.
23. Y. Yih and A. Thesen. *Semi-Markov Decision Models for Real-time Scheduling*. Research memorandum. School of Industrial Engineering, Purdue University, 1991.
24. J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, Sept. 2005.
25. J. Yu, R. Buyya, and C. K. Tham. Cost-based scheduling of scientific workflow application on utility grids. In *Proceedings of the First International Conference on e-Science and Grid Computing*, E-SCIENCE '05, pages 140–147, Washington, DC, USA, 2005. IEEE Computer Society.