Impact of Variable Priced Cloud Resources on Scientific Workflow Scheduling

Simon Ostermann and Radu Prodan

Institute of Computer Science, University of Innsbruck,
Technikerstr. 21a, 6020 Innsbruck, Austria

Abstract. We analyze the problem of provisioning Cloud instances to large scientific workflows that do not benefit from sufficient Grid resources as required by their computational requirements. We propose an extension to the dynamic critical path scheduling algorithm to deal with the general resource leasing model encountered in today's commercial Clouds. We analyze the availability of the cheaper and unreliable Spot instances and study their potential to complement the unavailability of Grid resources for large workflow executions. Experimental results demonstrate that Spot instances represent a 60% cheaper but equally reliable alternative to Standard instances provided that a correct user bet is made.

Keywords: Cloud computing, Grid computing, Spot instances, Scheduling, Scientific workflows, Performance, Cost.

1 Introduction

From the rather broad amount of definitions and interpretations of the term Cloud computing, the scientific computing community mostly focuses on the Infrastructure as a Service (IaaS) interpretation characterized by leasing of computation, storage, message queues, databases, and other raw resources from specialized providers. In this context, scientific workflows emerged in the last decade as a highly successful paradigm for programming loosely-coupled high-performance computing infrastructures such as computational Grids and Clouds. In this paper, we study the possibility of extending Grid infrastructures with IaaS Cloud resources to improve the execution of large workflow applications that do not have sufficient Grid resources available for their computational demands. We design an extension to the *Dynamic Critical Path (DCP)* algorithm which demonstrated in previous work [1] better results compared to other existing heuristic strategies for most types of workflows and independent of their size, in particular when the resource availability frequently changes or when using dynamic and unreliable heterogeneous resources. We build such an environment by using a combination of a set of cluster resources offered by the Austrian Grid, complemented on-demand by "pay-as-you-go" Cloud resources offered by Amazon EC2. Besides the Standard instances rented at a fixed price per hour, Amazon gives the possibility to bet on unused resources called *Spot instances* (SIs) and rent them at variable prices with no reliability guaranteed. To support workflow execution in such hybrid environment, we extend the DCP algorithm in three directions: (1) *Rescheduling* deciding when to start the Cloud

C. Kaklamanis et al. (Eds.): Euro-Par 2012, LNCS 7484, pp. 350–362, 2012.

instances to complement the unavailability of free Grid resources; (2) *Cloud choice* determining the type and maximum amount of Cloud resources to be provisioned, and the price bet for SIs; (3) *Prescheduling* trying to minimize the impact of the scheduling overhead in case of a large number of activities. We analyze the *Spot prices* offered by Amazon EC2 and their impact to the overall workflow execution time. We find that SIs have good potential to improve the workflow execution with a significant cost reduction for longtime usage, provided that a correct price bet is made.

2 Model

2.1 Resource Model

We adopt the resource model of Amazon EC2 that offers different resources called instances of different types, where we shown the three used ones in Table 1. The processor speed of these instances is quantified with a metric called *Elastic Compute Unit (ECU)* equivalent to the speed of an Opteron 2007 processor with approximately 1.2Ghz. Additionally, there are three pricing models for renting these resource:

- *Standard instances* let the customer pay for compute capacity by the hour with no long-term commitments.
- *Spot instances (SIs)* allow customers to bid on unused Amazon EC2 capacity and run those instances for as long as their bid exceeds the current spot price. The spot price changes periodically based on supply and demand, and customers whose bids exceed/meet the spot price gain/loose access to the SIs;
- *Reserved instances* give customers the option to make a one-time payment for each instance they want to reserve for one or three years and receive in-turn a significant discount on the hourly charge.

We analyzed first the Reserved instance prices which are cheaper than the Standard instances if their usage is in the interval of $[167.3, 173.1]$ days for one year reservations, and $[252.5, 265.2]$ days for three year reservations. Since this high utilization requirement does not match our execution scenarios characterized by occasional execution of experimental workflow sets, we concentrate our work on the SIs as a cheap and more interesting alternative (see Section 4).

2.2 Application Model

We focus in this paper on scientific workflow applications that can benefit from additional Cloud resources if there are no sufficient Grid resources to support their computational requirements. We model a scientific workflow $WS = (AS, DS)$ as a set AS of

Table 1. Overview of used Amazon EC2 Linux resources and prices for the *US-East* area as of 1.2.2012; Spot prices are averaged over the last 12 months

Name	ECUs (Cores)	RAM [GB]	Archi. [bit]	I/O Performance	Disk [GB]	Standard Cost [$/h]	Spot price [$/h]	Reserved Cost [$/h]	Reservation Cost [$/j, $/3j]
m1.large	4 (2)	7.5	64	High	850	0.34	0.2124	0.12	910, 1400
c1.xlarge	20 (8)	7.0	64	High	1,690	0.68	0.3151	0.24	1820, 2800
cc1.4xlarge	33.5 (8)	23	64	10 Gigabit	1690	1.30	0.6797	0.56	4290, 6590

legacy codes called *activities* interconnected in a directed acyclic graph through control flow and data flow dependencies DS. With no loss of generality, we assume that the workflow has one initial and one final activity which has no pre- or successors. As most legacy applications do not support checkpointing, we assume that this support is not available. This restriction means that, when a SI is terminated, the intermediate results of the currently running computational activities are also lost.

To model and estimate the execution time of workflow activities on Grid/Cloud resources, we use a simple performance modeling and prediction service tuned for our pilot applications that we presented in [2]. We use the benchmarks presented in [3] to model the performance of EC2 instances.

2.3 Dynamic Critical Path Algorithm

Dynamic Critical-Path Scheduling (DCP) is a static scheduling algorithm for allocating task graphs to fully connected multiprocessors [4], which demonstrated in previous work [1] better results compared to other existing heuristic strategies for most types of workflows and independent of their size, in particular when the resource availability frequently changes or when using dynamic and unreliable heterogeneous resources. The algorithm is based on the dynamic calculation of the *critical path* (CP) in each step. The CP is defined as the set of interconnected activities from the initial to the final activity with the maximum aggregated computation and communication costs. The workflow could achieve a minimum execution time if the CP is scheduled on the resources that deliver its earliest completion time (ECT) and all other activities are executed in parallel to the CP.

Algorithm 1. DCP algorithm.

Require: $W = (AS, DS)$: workflow application, RS: resource set
Ensure: : Schedule W on RS
1: Compute *EST* **and** *LST* for all activities on all resources
2: **while** Not all activities are scheduled **do**
3: Select $A \in AS : \min(LST_A - EST_A)$ **and** $\min(EST_A)$
4: **if** $LST_A = EST_A$ **then**
5: $onCP \leftarrow true$
6: **else**
7: $onCP \leftarrow false$
8: **end if**
9: $SelectResource(A, onCP, RS)$
10: Update EST_A **and** $LST_A, \forall A \in AS$
11: **end while**

Algorithm 1 briefly outlines the DCP pseudocode that takes a workflow as input parameter and maps its activities onto the available resources for execution. First, it calculates the *earliest possible start time* (EST) and *latest possible start time* (LST) for all activities on all resources (line 1), which allows the identification of the CP. Then, the activity A with the minimum EST_A and the minimum start flexibility ($LST_A - EST_A$) is chosen for scheduling (line 3). The activities for which $LST_A = EST_A$ are on the critical path and will therefore be scheduled first. Each activity is scheduled on the resource delivering its ECT such that the LST and EST are satisfied (line 9). The *SelectResource* function (explained in Section 5.1) needs to consider if the activity is on the CP, in which case it reconsiders the already scheduled activities as the CP may have changed. This is indicated by the boolean $onCP$ variable, set to true if $LST_A = EST_A$ (lines 4–8). Depending on the scheduled activities, the EST and LST are recalculated for all activities and the algorithm repeats until all activities are scheduled (lines $10 - 11$).

3 Related Work

There are a number of important projects showing a growing interest in Cloud computing in the scientific and open source communities. The Nimbus [5] package provides a scientific Cloud middleware deployed in an informal group of four university Clouds called Science Cloud. Hadoop [6] is a toolkit for distributed computing allowing map-reduce applications to be developed and executed in a complete tool chain that supports Cloud resources and Grids. A commercial open-source implementation of a Cloud middleware compatible with EC2 is provided by Eucalyptus [7].

In [8], the impact of checkpointing when using unreliable Cloud resources is analyzed. However, scientific applications often consist of legacy codes that do not the support checkpointing mechanism needed for such an approach. Our solution shows that there is potential for using unreliable SIs even without checkpointing.

Other approaches [9] used Cloud resources to extend clusters when more requests need to be processed than the cluster can handle. A simple load balancer submits jobs either to the cluster or to the Cloud, while in our approach the Cloud instances are completely integrated into the resource pool.

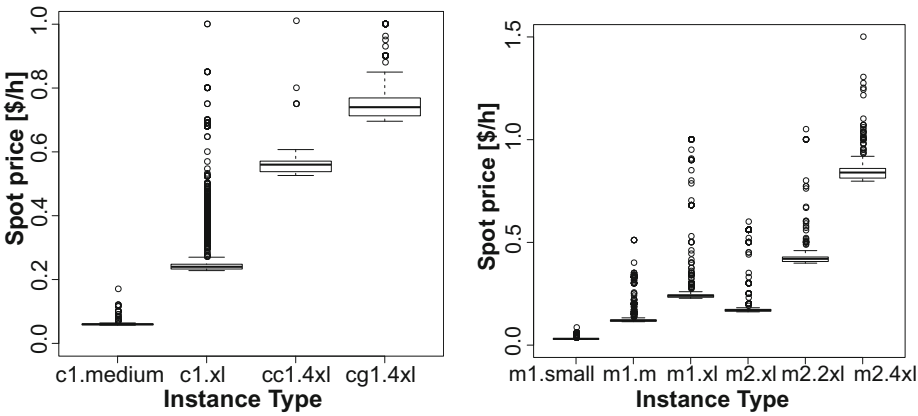
In [10], several BPEL extensions for using Cloud resources in cases of peak load situations are proposed. The approach does not optimize the use of the Cloud resources, nor is well-suited for massively parallel applications, as one of the constrains of the proposed extensions is that execution servers may handle multiple requests simultaneously. In contrast, scientific applications are mostly designed to utilize dedicated resources.

The work presented in [11] is comparable to our approach, but focuses on fault tolerant scheduling that does not optimize the schedule for Cloud use and cost. Our approach is optimizing these metrics with support for cheaper SIs.

An extension of the Torque job manager to add Cloud resources to clusters is presented in [12]. Our approach could interoperate with such job manager by integrating cluster, Clouds and Grid in one single computing environment. Similar work has been done in [13] where clusters can dynamically extend their capacity with Cloud resources upon peak usage to meet given service level agreements. This gives to resource owners control over the Cloud usage and gives no transparency of the cost to the enduser, like the scheduler we proposed in this paper.

4 Spot Price Analysis

Most Cloud providers offer resources for a fixed price on an hourly basis. Since recently, Amazon EC2 introduced SI with a market-oriented dynamic pricing to better utilize the unused resources from their resource pool and we analyze them as a case study for variable prices in general. SI have a dynamic price that changes periodically based on their supply and customer's demands and bids. If the Spot price exceeds the user's bet, the instance is automatically terminated and the user does not have to pay for the last uncompleted billing interval. In the following, we analyze the Spot prices for all available instance types for the last 12 months. Since 7.2011, SI have different prices for the different availability zones of a region. Nevertheless, the data we collected until now does not show any significant difference from the overall pricing. The shown



(a) Newer instance types.

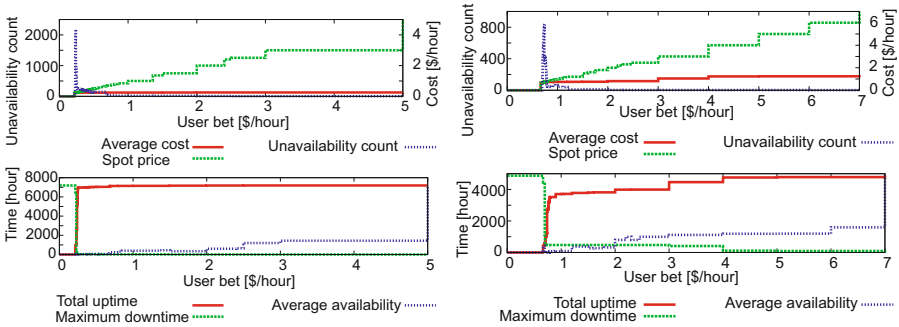
(b) Original (older) instance types.

Fig. 1. EC2 Spot price analysis for the time period 22.1.2011 – 1.2.2012

methods to analyze the impact of the user bet on the reliability could be applied to other providers once they introduce similar resource models.Spot prices can be queried using the command line tools provided by Amazon EC2. Users accessing the EC2 Cloud can request Spot price information of any instance type, region, operating system and availability zone.

Figure 1 shows box-plots of the different instance types and their prices. We removed some of the outliers in Figure 1a, as the `cg1.4xlarge` instance had one single Spot price of $7.0 and two to four of $2.0, $2.45, $3.0, $4.0, $5.0, $6.0 and $7.0 per hour, while all other prices were lower than $1.00. We do not show the outliers in Figure 1b for the `m2.4xlarge` instance ($2.0 and $3.0 per hour), which only occurred once in the monitored time period. As the markers for the percentile of the dataset show, the Spot prices are stable for the `m1.small`, `c1.medium`, `m2.xlarge`, and `m2.2xlarge` instances. For the other six instance types, the price ranges are larger as shown by the visible whiskers, especially for the six `xlarge` types (except `m2.xlarge`). Not only the price fluctuations are important, but also the time periods for which the prices are valid, which we analyze in Figure 2. For a user who wants to use a SI for several hours or days, not only the maximum price to be payed is of interest, but also the average price resulting from the overall usage which approximates the cost to be payed. We analyzed the following metrics for all instance types:

- *user bet* is the price that a user will to pay for the execution of one SI for one hour;
- *average cost* is the hourly price a user has to pay for the SI if it were executed using his bet averaged over the complete timespan of our analysis;
- *maximum Spot price* is the maximum SI cost encountered still below the user bet;
- *unavailability count* represents the total number of shutdowns that occur when trying to run a SI with the user bet for the whole time period;
- *total uptime* sums the total time an instance was executed over the analyzed period with the given user bet;



(a) The `c1.xlarge` instance type.

(b) The `cg1.4xlarge` instance type.

Fig. 2. Instance uptime and average price analysis from 22.1.2011 until 1.2.2012

- *maximum downtime* is the longest time period where the market price of a SI was above the user bet;
- *average availability* is the time between each enforced shutdown averaged across the unavailability count.

Due to space limitations, we only present the two most interesting instances: `c1.xlarge` as the mostly used instance and `cg1.4xlarge` as the most expensive and fluctuating.

Figure 2a shows that for the `c1.xlarge` instance, the user bet has an insignificant influence on the average price to be paid for its long-term usage. The top chart shows that for a low bet close to $0.22 per hour, the unavailability count can reach a peak value of 2200 which rapidly decreases when increasing the bet. We conclude that the user shall be more generous in his bet to obtain a reliable infrastructure. The maximum Spot price follows a step function showing jumps in the market prices (for example there were no prices between $1.0 and $1.36). The bottom chart illustrates that the total uptime grows as soon as the lowest market price of 0.22 is met and reaches a value of 98% at $0.252 per hour. The maximum downtime never took longer than 32 hours for a user bet over $0.24 per hour. The average availability grows to a value of 125 hours for a user bet of $0.68 per hour and raises further except for user bets between [$1.5, $2.0] per hour for which it slightly decreases as the unavailability count grows from 16 to 20.

In Figure 2b we analyze the new `cg1.4xlarge` instance type, currently only available in one of the eight regions of EC2 and since 6.4.2011. The top chart shows that the average cost increases by 14% for the analyzed period if the user bet is increased from $3.0 to $4.0 per hour. The maximum SI cost is increasing again stepwise, as in the case of the `c1.xlarge` instance. For a user bet close to $0.74 per hour, the unavailability count reaches a peak value of 833 and decreases slower than for the `c1.xlarge` instance when increasing the bet. We conclude that the user shall be more generous with this type and bet about $1.2 per hour to obtain a reliable SI. The bottom chart shows that the user bet has a higher influences on total uptime as market price fluctuations are higher. The maximum downtime for user bets up to $4 per hour is above 407 hours and

reduces to 113 hours with bets over $4. The average availability is again related to the unavailability count and fluctuates up to a bet of $2.45 per hour where it reaches 998 hours. Nevertheless, it shows significants improvement starting from $2.0 per hour.

The values we were able to gather from this analysis are used to estimate the recommended user bet for SI that are required to get reliable results.

5 Dynamic Critical Path for Clouds

We extend the DCP full-ahead scheduling algorithm for minimizing the workflow makespan in a dynamic environment that uses Cloud SIs if the Grid resources are not sufficient for executing large workflows. If the Spot price increases to a higher value than, the user bet, a rescheduling action is triggered to move activities from the SIs that are terminated. The Spot price is then monitored every minute using a pull mechanism until the market price is below the user bet and a new instances can be requested. A *price buffer* is added to the market price to avoid constant rescheduling when the price is fluctuating close to the user bet. If the new price is lower, the scheduler will reschedule the workflow again, compare the expected execution time and cost with the previous schedule, and use the better mapping for the activities that have not been yet submitted.

5.1 DCP-C Algorithm

The DCP-C algorithm (see Algorithm 2) is based on the original DCP presented in Section 2.3. We extend in this paper the *SelectRessource*$(A_i, onCP)$ function called by the main loop for mapping each activity A_i, where the *onCP* boolean variable indicates whether the activity is on the CP. For each available resource (line 3), the EST is first calculated using Algorithm 3 (lines 4–7) which considers whether the activity is on the CP or not. Afterwards, the child with the minimum $LST - EST$ difference is chosen (line 9) and a slot that fulfills its EST and LST is searched (line 10). The resource that delivers the minimum EST sum of the two activities is then chosen for scheduling.

We extended the *FindSlot* function for finding a free resource slot that satisfies the EST and LST constraints of an activity to support Cloud resources, in particular SI targeted by our work. We achieved this by adding a new resource type called *NewCloud*. If a task is mapped to a new Cloud instance, we add the startup latency of this resource (i.e. premeasured virtual machine deployment and boot times) to the estimated EST (lines 2–4), which we determine based on our previous benchmark analysis work [3]. The scheduler then looks for a resource slot that allows the execution of the activity A_i within its range $[EST, LST]$ (lines $6 - 8$) and returns an infinite start time if none is available. If the task is on the CP, the scheduler needs to consider the already mapped activities and maybe even delay some of them if the CP has been changed (lines $9 - 11$). Otherwise, the scheduler needs to check that no dependencies are violated by the new slot (line 12). Finally, the function returns the start time of the found timeslot (line 13).

In the following, we present three simple optimizations aiming to reduce the makespan of large workflows running on Grid resources complemented with on-demand dynamic SIs: rescheduling, Cloud choice, and prescheduling.

Algorithm 2. *SelectResource* function.

Require: : A_i: workflow activity; $onCP$: flag set true if $A_i \in CP$; RS: resource set;
Ensure: : Schedule A_i to one resource in RS
1: $S \leftarrow null$
2: $EST2 \leftarrow \infty$
3: **for all** $R \in RS$ **do**
4: $EST_i \leftarrow FindSlot(A_i, R, false)$
5: **if** $EST_i = \infty$ and $onCP$ **then**
6: $EST_i \leftarrow FindSlot(A_i, R, true)$
7: **end if**
8: **if** $EST_i \neq \infty$ **then**
9: Select $A_c \in successor(A_i)$: $\min(LST_c - EST_c)$
10: $cST \leftarrow FindSlot(A_c, R, false)$
11: **if** $cST + EST_i < EST2$ **then**
12: $S \leftarrow R$
13: $EST2 \leftarrow cST + EST_i$
14: **end if**
15: **end if**
16: **end for**
17: Schedule A_i to S

Algorithm 3. *FindSlot* function.

Require: : A: workflow activity; R: resource; $onCP$: flag set true if $A_i \in CP$;
Ensure: : start time of A_i on R;
1: Calculate EST and LST of A on R
2: **if** $R = NewCloud$ **then**
3: $EST \leftarrow EST + CloudLatency$
4: **end if**
5: $(Start, End) \leftarrow Get{-}Slot(A, R, EST, LST)$
6: **if** $[Start, End] \not\subset [EST, LST]$ **then**
7: **return** ∞
8: **end if**
9: **if** $onCP$ **then**
10: Check if CP has changed **and** activities no longer on CP need delay
11: **end if**
12: Check if schedule does not violate dependencies
13: **return** $Start$

5.2 Rescheduling

There may appear situations during workflow execution when additional shared Grid resources become available, or when Spot prices change so that some instances may get terminated or additional ones started. This is detected by the scheduler that continuously monitors the Spot price, compares it with the user bet, and triggers a rescheduling operation to adjust the mapping of activities to the new infrastructure configuration. To perform rescheduling, the scheduler marks each activity as not scheduled except for the finished and currently running ones, and maps unexecuted subworkflow again using the proposed DCP-C algorithm. The user sets a price buffer that adds additional inertia to the mechanism to keep the number of rescheduling operations within reasonable limits.

5.3 Cloud Choice

In our model, an important task of the scheduler is to dynamically complement the Grid infrastructure with additional Cloud resources during runtime if this presents potential for improving the workflow execution. In our case, this happens when the amount of Grid resources are insufficient for executing large workflow parallel regions that need to be serialized. The scheduler detects this situation when there are no free slots on the resource an activity gets mapped to, which results in an increase in the planed starttime. In selecting a new Cloud instance, the scheduler takes two important decisions: (1) the instance type to lease from the Cloud provider, and (2) the number of such instances.

Selecting the instance type requires analysis of three important metrics: resource speed, cost, and reliability. If the total budget available for a workflow is known, the scheduler estimates the overall makespan on the Grid resources, and approximates the hourly budget required. Then, it uses this information to decide which instance type to use (representing the speed and cost parameters), including whether it should be a SI or a Standard instance (representing reliability and cost). Based on our trace data analyzed

in Section 4, the scheduler will prefer SIs as long as their current market price is below the Standard price. The Spot prices are in average up to 60% lower than the regular costs (see Table 1) allowing cheaper executions with a similar reliability. To keep the complexity of the DCP-C algorithm low, this functionality is hidden in the *CloudLatency* parameter when calculating the EST on the *NewCloud* (line 3 in Algorithm 3). We quantify the speed of a Grid resource or Cloud instance in *Elastic Compute Units* (ECU), which we compute based on our previous Cloud benchmarking work [3]. Using this unit, we compare the resource speeds and predict the activity execution times for our computation-intensive workflows. When selecting the Cloud instance type, we use the maximum number of parallel activities and the total amount of available Grid resources. Since Cloud instances are only offered in bulks of cores, we use a minimum resource utilization input by the user to estimate the amount of Cloud instances needed.

5.4 Prescheduling

The DCP-C algorithm has an $O(N^3)$ complexity, where N is the number of activities, and has a non-negligible execution time for workflows with a large number of activities. To minimize the impact of this overhead on the execution time for large workflows (above 1000 activities), the scheduler does first an immediate mapping of the initial activities (most likely to be on the CP) on the fastest resources, before calling the DCP-C algorithm in a rescheduling-similar manner. With this hybrid approach we are able to obtain good scheduling results combined with a reduced scheduling overhead.

6 Evaluation

We designed and implemented the methods presented in this paper in the ASKALON environment [14] designed to support scientists in modeling, scheduling, and executing scientific workflows in Grid and Cloud infrastructures. The Cloud support is enabled through a back-end interface to the Eucalyptus middleware, compliant with the EC2 interface. To support a large number of experiments required to validate new methods, we interfaced the ASKALON enactment engine to the GroudSim [15] simulator that enables deterministic simulations of applications comprising job executions, file transfers, cost calculations, and background load on of Grid and Cloud computing infrastructures.

We used GroudSim to simulate three sites of the Austrian Grid environment and the instance types offered by Amazon EC2 (see Tables 1 and 2). We assumed a standard Amazon account which allows us to acquire a maximum of 20 instances, which results in a total of 160 cores when using the `cc1.4xlarge` instance type. We used the ASKALON environment interfaced to GroudSim to simulate two real-world scientific workflows from the material science and hydrological domains with different structure, number of activities, and computational requirements. Each workflow is characterized by a parameter x defining the "parallelization size", which determines the total number N of workflow activities. We used this parameter to simulate workflows of different sizes (small to very large), which we executed first in a pure Grid environment and then by using on-demand Cloud resources.

Table 2. Simulated Austrian Grid resources

Grid site	Location	Cores	Speed [ECU]
karwendel	Innsbruck	80	2.5
altix1.uibk	Innsbruck	16	1.5
altix1.jku	Linz	64	2.0

To model and estimate the execution time of workflow activities on Grid/Cloud resources, we use a simple performance modeling and prediction service tuned for our pilot applications that we presented in [2]. To better quantify the computational performance of one ECU we use the performance benchmarks and models from previous work [3]. The average size of a file transfer in our workflows is of around 100 kilobytes, which is insignificant compared to the total computation time. Staging files into the Cloud is mostly free of charge, while staging files out of the Cloud cost around $0.12 per gigabyte, resulting in a negligible influence on the aggregated workflow cost. In all experiments, we selected from the historical data from EC2 an interval with high SI prices to study the impact of instance termination and the resulting rescheduling.

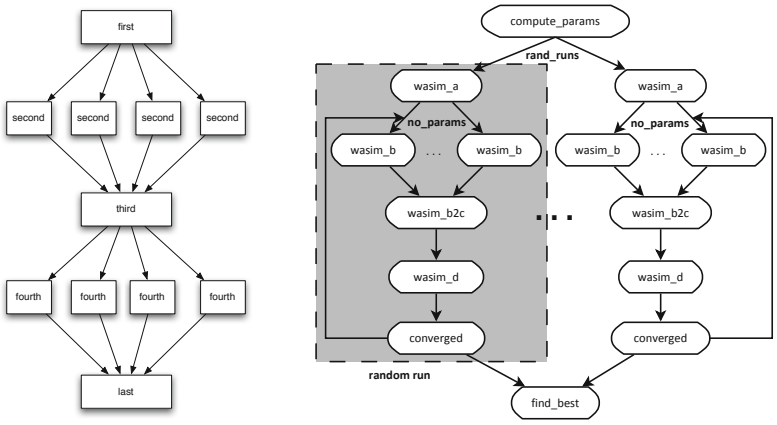
6.1 Wien2k

Wien2k is a material science workflow for performing electronic structure calculations of solids.The Wien2k workflow contains two parallel sections of size x, with sequential synchronization activities in between (see Figure 3a). The total number of activities in a Wien2k workflow is: $N_{wien2k} = 2 \cdot x + 3$.

Figure 4a shows our simulation results for scheduling and executing Wien2k workflow with different parallelization sizes. The chart first shows that using additional Cloud instances brings significant improvements in the execution time of this workflow. The faster instance types are selected by the scheduler and the more cores are added to the resource pool, the lower the execution time of the workflow gets. Using user bets of $0.35 for `m1.large`, $0.68 for `c1.xlarge` and $1.01 for `cc1.4xlarge` instances gives us a 99% reliability according to our analysis in Section 4 (meaning a value in the area of high availability with acceptable average prices). This means that there will be no reschedules in the execution on the SIs resulting in identical curves as when using the Standard instances. The cost for using SIs is about 60% cheaper than using the Standard instances for the same executions (see Table 1). The figure also shows that choosing lower user bets of $0.125 for `m1.large`, $0.25 for `c1.xlarge` and 0.58 $ for `cc1.4xlarge` instances closer to the Spot price increases the completion time due to SI unavailability and rescheduling operations (e.g. for the `cc1.4xlarge` instances with a parallelization size of 388). In the worse case, the same execution times as when using Grid resources only are obtained, meaning that the SIs have been paid with no benefit. Smaller workflows with $x < 200$ or 80 hours of runtime finish before any SI gets terminated which produces overlapping lines in the chart.

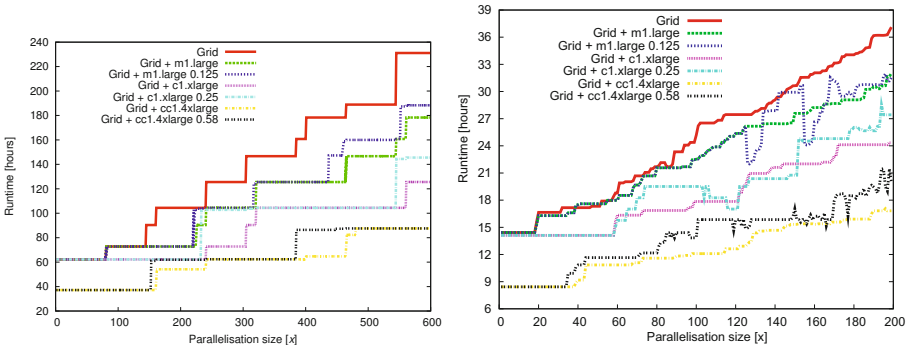
6.2 Invmod

Invmod [16] is a hydrological application that uses the Levenberg-Marquardt algorithm to minimize the least squares of the differences between the measured and the simulated runoff for a determined time period. The Invmod workflow displayed in Figure 3b



(a) Wien2k with $x = 4$. (b) Invmod with $x = rand\_runs$.

Fig. 3. Scientific workflows



(a) Wien2k (b) Invmod

Fig. 4. Execution times of different parallelization sizes and instance types

consists of two levels of parallelism: (1) the outermost parallel loop consists of a number of random runs x (parallelization size) that perform a local search optimization (in a sequential loop) starting from a random initial solution; (2) alternative local changes are examined for each calibrated parameter in parallel in the inner nested parallel loop. The total number of jobs in an Invmod workflow is: $N_{invmod} = 13 \cdot x + 2$.

Figure 4b shows our experiments results with the Invmod workflow. Since this workflow has a higher complexity than Wien2k, the benefits are smaller when slower `m1.large` instances are added to the resource pool. On the other hand, the impact of the cluster resources `cc1.4xlarge` is higher, as they outperform the available Grid hardware by their speed and core amount ($20 \cdot 8 = 160$ cores). Similar to the Wien2k workflow, a correct user bet for SIs can bring similar performance results with over 60% cost reduction (see Table 3). The `m1.large` executions show that for some parallelization sizes ($x = 140$ or $x = 150$) the performance of the workflow running on SIs

that get terminated can be as low as a Grid only execution. SIs can be again detrimental when setting a too low user bid for all three instance types as compared to Standard instances. In rare cases, rescheduling triggers an improvement in execution time due to the better mappings for the remaining subworkflows obtained by the CP heuristic.

7 Conclusion

The choice of the correct Cloud instance is critical when running scientific application in the Cloud. While cheaper resources might look attractive, their slow characteristics degrade the performance in such a way that the overall execution time is not significantly improved and in the worst case even decreased. In this paper we analyzed the potential of using SIs for improving the performance of real-world scientific workflows that suffer from the insufficient Grid resources compared to their computational demand. We first collected and analyzed the characteristics of the Spot prices over a period of 12 months and found out that they are quite stable for the different instance types offered by Amazon. Furthermore, SIs offer over 99% availability when the user makes a correct bet of $0.35 for `m1.large`, $0.68 for `c1.xlarge` and $1.01 for `cc1.4xlarge` instances with a average price far below the standard price. Then, we extended the Dynamic Critical Path algorithm for Cloud environments using three simple extensions: rescheduling, Cloud choice, and prescheduling. We learned that SI can bring significant improvements similar to the Standard instances to the execution of scientific workflows in hybrid Grid-Cloud environment, provided that the correct user bet is made. Cost-wise, SIs can bring over 60% reduction in costs if correctly used. With the upcoming Spot price per availability zone, there will be even more room for improving the execution cost by dynamically choosing the cheapest zone for the end-user.

Table 3. Cost comparison when using Spot and Standard instances

Workflow	Size [x]	Instance Type	Cost [$]	Spot price [$]	saved [%]
wien2k	400	m1.large	439.28	167.743	61.81
wien2k	200	cc1.4xlarge	1771.2	648.818	63.37
invmod	140	c1.xlarge	171.36	64.726	62.23
invmod	160	cc1.4xlarge	385.6	154.047	60.05

Acknowledgment. The research was funded by the Austrian Science Fund (FWF): TRP 72-N23 and the Standortagentur Tirol: RainCloud.

References

1. Rahma, M., Venugopal, S., Buyya, R.: A Dynamic Critical Path Algorithm for scheduling Scientific Workflow Applications on Global Grids. In: eScience, pp. 35–42. IEEE Computer Society (2007)
2. Nadeem, F., Yousaf, M., Prodan, R., Fahringer, T.: Soft benchmarks-based application performance prediction using a minimum training set. In: e-Science. IEEE Computer Society Press (2006)
3. Iosup, A., Ostermann, S., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. IEEE TPDS 22(6), 931–945 (2011)

4. Kwok, Y.-K., Ahmad, I.: Dynamic Critical-Path Scheduling: An effective Technique for allocating Task Graphs to Multiprocessors. IEEE TPDS 7(5), 506–521 (1996)
5. Keahey, K., Freeman, T., Lauret, J., Olson, D.: Virtual Workspaces for Scientific Applications. In: Scientific Discovery through Advanced Computing, Boston (June 2007)
6. Apache, Apache Hadoop Project develops open-source Software for Reliable, Scalable, Distributed Computing (May 2010), http://hadoop.apache.org/
7. Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D.: Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. UCSB Computer Science. Tech. Rep. 2008-10 (2008)
8. Sangho, Y., Kondo, D., Andrzejak, A.: Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In: CLOUD, pp. 236–243. IEEE (2010)
9. Assuncao, A.C.M., Buyya, R.: Evaluating the Cost-Benefit of using Cloud Computing to Extend the Capacity of Clusters. In: HPCC. ACM (2009)
10. Dörnemann, T., Juhnke, E., Freisleben, B.: On-demand Resource Provisioning for BPEL Workflows using Amazon's Elastic Compute Cloud. In: CCGrid, pp. 140–147. IEEE Computer Society (2009)
11. Ramakrishnan, L., Koelbel, C., Kee, Y.-S., Wolski, R., Nurmi, D., Gannon, D., Obertelli, G., YarKhan, A., Mandal, A., Huang, T.M., Thyagaraja, K., Zagorodnov, D.: Vgrads: enabling e-Science Workflows on Grids and Clouds with Fault Tolerance. In: SC. ACM (2009)
12. Marshall, P., Keahey, K., Freeman, T.: Elastic Site: Using Clouds to Elastically Extend Site Resources. In: CCGrid, pp. 43–52. IEEE (2010)
13. Blanco, C.V., Huedo, E., Montero, R.S., Llorente, I.M.: Dynamic Provision of Computing Resources from Grid Infrastructures and Cloud Providers. In: GPC Workshops, pp. 113–120. IEEE Computer Society (2009)
14. Fahringer, T., Prodan, R., Duan, R., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.L., Villazón, A., Wieczorek, M.: ASKALON: A Grid application development and computing environment. In: GRID, pp. 122–131. IEEE (2005)
15. Ostermann, S., Plankensteiner, K., Prodan, R.: Using a New Event-based Simulation Framework for Investigating Different Resource Provisioning Methods in Clouds. Scientific Programming Journal 19(2-3), 161–178 (2011)
16. Cullmann, J., Mishra, V., Peters, R.: Flow analysis with WaSiM-ETH - model parameter sensitivity at different scales. Advances in Geosciences 9, 73–77 (2006)