Comparing Provisioning and Scheduling Strategies for Workflows on Clouds

Marc E. Frincu, Stéphane Genaud, Julien Gossa

Comparing Provisioning and Scheduling Strategies for Workflows on Clouds

Marc E. Frincu, Stéphane Genaud and Julien Gossa
ICube-ICPS – UMR 7357, Université de Strasbourg, CNRS
Pôle API 300 Blvd S. Brant, 67400 Illkirch France
Email: {frincu,genaud,julien.gossa}@unistra.fr

***Abstract*—Cloud computing is emerging as a leading solution for deploying on demand applications in both the industry and the scientific community. An important problem which needs to be considered is that of scheduling tasks on existing resources. Since clouds are linked to grid systems much of the work done on the latter can be ported with some modifications due to specific aspects that concern clouds, e.g., virtualization, scalability and on-demand provisioning. Two types of applications are usually considered for cloud migration: bag-of-tasks and workflows. This paper deals with the latter case and investigates the impact virtual machine provisioning policies have on the scheduling strategy when various workflow types and execution times are used. Five provisioning methods are proposed and tested on well known workflow scheduling algorithms such as CPA, Gain and HEFT. We show that some correlation between the application characteristics and provisioning method exists. This result paves the way for adaptive scheduling in which based on the workflow properties a specific provisioning can be applied in order to optimize execution times or costs.**

***Keywords*-workflow scheduling; virtual machine provisioning; cloud computing**

I. Introduction

The emergence of cloud computing caught the attention of not only the industry but academia as well. Researchers are interested both in designing and applying models, standards and platforms to on-demand pay-per-use infrastructures.

They usually deal with two kinds of experiments, those consisting of many independent tasks, i.e., bag-of-tasks (BoT), and those in which tasks are interconnected through dependencies, i.e., workflows. Workflows can be either deterministic – the execution path can be determined a priori, which is usually the case of directed acyclic graphs (DAGs) – or non-deterministic – the execution path is usually determined at runtime and consists of loop, split and join constructs [1]. Commercial cloud applications can be divided similarly: single service oriented – e.g., weather, stock exchange – or workflow oriented – e.g., bank transfers, online reservations, social websites.

One of the problems involving experiments on grids and clouds alike is that of tasks-to-resources mapping. The issue has been widely addressed for grids while clouds only recently gained attention. For the latter the problem is three folded since it requires scheduling on three levels: (1) finding the appropriate Physical Machines (PMs) for a set of Virtual Machines (VMs); (2) determining the best provisioning scheme for the (VMs) on them; and (3) scheduling the tasks on the VMs. Depending on the level of access a service provider has to the infrastructure it can influence one or more of them.

Many grid Scheduling Algorithms (SA) for BoT have been extended and adapted to clouds [2], [3], [4], [5]. These papers show the various impact VM provisioning policies have on the same task scheduling method. These results add complexity to the scheduling problem since providers need to be concern with VM provisioning as well as with task scheduling.

The problem of workflow scheduling has mostly tackled the aspect of extending previous grid SA and has ignored, to our best knowledge (cf. Sect. II), to study the impact VM provisioning has on the scheduling policy. The aim of this paper is to investigate this aspect and to study its impact when considering cost and time constraints.

We therefore propose several VM provisioning policies and test them on known workflow SAs with various execution time patterns. We address CPU intensive cases and show that the SA outcome is linked to the workflow structure as well as to the provisioning method.

The rest of the paper is structured as follows: Section II depicts some of the main results in the field of cloud SAs for BoTs and workflows. Section III depicts the main VM provisioning policies (cf. Sect. III-A) and task allocation strategies (cf. Sect.III-B) used in our tests. The experimental set up is presented in Sect. IV. Results are discussed in Sect. V while the conclusions and future work are addressed in Sect. VI.

II. Related Work on Cloud Scheduling

Cloud computing as an evolution of grid computing has brought with it many of its predecessor problems, including those related to scheduling. Much work has been done to adapt existing algorithms used in grids for clouds. These include both BoT and workflow oriented. The scheduling problem is known to be $\mathcal{NP} - complete$ while the *bin packing problem* of fitting multiple VMs on a PM is known to be $\mathcal{NP} - hard$ [6].

One of the first works to show the cost of doing science on the clouds is [7] where three cloud migration scenarios are depicted: (1) use clouds sporadically to enhance the local infrastructure, (2) deploy the entire application on the cloud

but keep the data locally, and (3) deploy both data and application on the clouds.

Most of the commercial clouds use simple allocation methods such as Round Robin (Amazon EC2[1]), least connections and weighted least connections (Rackspace[2]). One reason for this approach would be that providers target in general only load balancing for their PMs. Other simple SAs include Least-Load or Rotating-Scheduling [2].

Many SAs for clouds have been proposed for BoTs and there is some work [3], [4], [5] that analyses the impact of VM provisioning has on the SA. Tests have shown a dependency between the two which impacts both the makespan gain and the paid cost.

Regarding workflows most work has ignored the impact VM provisioning can have on the SA and focused on extending existing algorithms such as HEFT [8], CPA [9] and Gain [10]. Results in this direction include SHEFT [11] an extension of HEFT which uses cloud resources whenever needed to decrease the makespan below a deadline; CPA and biCPA [1] versions for determining the needed number of VMs a workflow requires; or Gain [10], [12].

In [13] several SA and VM provisioning strategies for workflows are presented. The policies rely on the first available resource to schedule the tasks. Some of the proposed policies are energy aware by relying on idle VMs instead of running new ones. The authors consider energy consumption and makespan as primary goals.

Liu et al. [14] propose three heuristics-based workflow SAs for instance-intensive schedules: Throughput Maximization Strategy, Min-Min-Average and Compromised-Time-Cost.

Scaling-Consolidation-Scheduling an algorithm for auto-scaling workflow applications is proposed in [12].

Byun et al. [15] propose a near optimal solution for minimizing costs based on task deadlines. They take advantage of the minimum makespan - deadline difference to extend a workflow's makespan and reduce its costs.

Other algorithms include auction based scheduling [16]; HCOC [17] which is based on the Path Clustering Heuristic (PCH) [18] and uses an approach similar to SHEFT when provisioning cloud resources; Particle Swarm Optimization [19], [20]; Genetic Algorithms [20]; and Ant Colony Optimization [20] methods.

A distinct category seems to be that of map-reduce workflows. These are highly parallel applications consisting of two phases, a map phase in which keys are generated based on map tasks and a reduce phase in which keys are read and results are produced. In [21] two algorithms for minimizing VM rent costs are proposed: List and First-Fit – sorts prices and the corresponding VMs are allocated to map and reduce tasks – and Deadline-aware Tasks Packing – uses the estimated deadline to schedule map tasks.

Despite the numerous solutions for workflow scheduling work we notice a lack in references to the dependability of the workflow schedule efficiency on the VM provisioning policy. A comparison between several workflow types and a hybrid private cloud + public cloud SA called HCOC is given in [17]. However the provisioning strategy apparently follows a one VM for every task approach. Since it has been proven that for BoTs the provisioning influences scheduling there is no reason to suspect that workflow would be any different. We thus require a way of knowing which VM provisioning and task scheduling to combine for given scenario classes.

Deelman et al. [7] underline the costs of deploying a scientific workflow on clouds using various approaches. The study shows the price needed to be paid when data intensive application are targeted for cloud deployment and does not offer a comparison among different scheduling strategies.

It is the intention of this work to show that for specific workflow types and goals – i.e., in this work we rely on cost and makespan and idle time – the efficiency of the scheduling technique relies on the provisioning strategy. Having a broad view on how several SAs behave with respect to certain platform and application properties is thus essential in determining which one is suited for the user goals.

III. Task Allocation and VM Provisioning Strategies

In this section we describe the main VM provisioning and task allocation strategies used in our experiments.

A. VM Provisioning

In this work VM provisioning refers to how VMs are allocated: reusing existing (idle) ones; renting a new one for new tasks or when the task execution time exceeds the remaining Billing Time Unit (BTU) [4] of a VM; considering or not boot time; etc.

As already shown for BoTs [4] how VMs are provisioned affects the *idle time*, *rent cost* and *makespan* of the schedule. In this work we study five simple provisioning policies.

Figure 1 exemplifies them on a simple CSTEM sub-workflow (cf. Sect. IV-B) consisting of one initial task and subsequent six tasks (cf. Fig. 1). The idle time is indicated by the dark *I*-marked rectangles while the default length of a BTU is represented by the rectangle marked BTU.

OneVMperTask: assigns a new VM to each task even if there remains enough idle time on another that could be used by the ready task.

StartParNotExceed: assigns a new VM to every initial workflow task. The rest of the tasks are scheduled sequentially on existing VMs – for a given task the VM with

[1]http://aws.amazon.com/ec2/ (accessed Dec 7th 2012)

[2]http://www.rackspace.com/blog/announcing-cloud-load-balancing-private-beta/ (accessed Dec 7th 2012)

the largest execution time is chosen – unless the BTU is exceeded and a new VM is rented for it.

StartParExceed: is similar to the previous one but no new VM is rented if a VM's BTU is exceeded – i.e., existing ones are used. If a single initial task exists this heuristic will schedule all workflow tasks in parallel

AllParNotExceed: assigns each parallel task to its own VM – existing or new. New VMs are added if the number of parallel tasks exceeds the number of VMs or if a task execution time exceeds the assigned VM's BTU. Sequential tasks are executed on the VM with the longest execution time – usually their (largest) predecessor.

AllParExceed: is similar to the previous but exceeding the BTU does not lead to new VMs being rent.

Since EC2 prices for on demand VMs follow the $cost_{BTU/core} \times \#cores$ formula, the last two strategies assume renting a new VM for each parallel task instead of using a multi-core VM. In an offline scenario the latter impacts only the global idle time not the makespan or cost.

It can be noticed that each provisioning strategy provides a different result in terms of allocated VMs, idle time, cost and makespan. The *OneVMperTask* and *StartParExceed* policies represent upper limits with regard to the cost respectively makespan. In addition, they lie on opposite sides when relating to idle time: *OneVMperTask* produces the largest idle time while *StartParExceed* gives neglectable values.

StartParNotExceed produces a slightly smaller makespan than *StartParExceed* but allocates more VMs and outputs a larger idle time.

The *AllParExceed* strategy exploits the task parallelism. It reduces makespan at task level and also costs since sequential tasks are allocated on the same VM. *AllParNotExceed* is similar but the resulted idle time is larger than that of *AllParExceed*. The efficiency of the two is however limited if little parallelism exists in the workflow. In that case they become *StartParExceed* or *StartParNotExceed*.

Depending on the task size and workflow structure *AllParNotExceed* and *StartParNotExceed* are methods which can provide different cost results from their counterparts. Section V will give more details on this aspect.

Overall the strategies that tend to allocate more VMs are better suited for tasks with large data dependencies where the VM should be as close as possible to the data. On the other hand these give large idle times resulting in a waste of budget. How these provisioning policies impact various workflow types is discussed in Sect. V.

B. Task Allocation

Task allocation refers to how tasks are mapped on VMs. The most common workflow SAs rely on *priority ranking*, *level ranking* or *clustering*. In this work we focus our attention on the first two methods. The first one lies at the foundation of the HEFT algorithm. For each task a rank

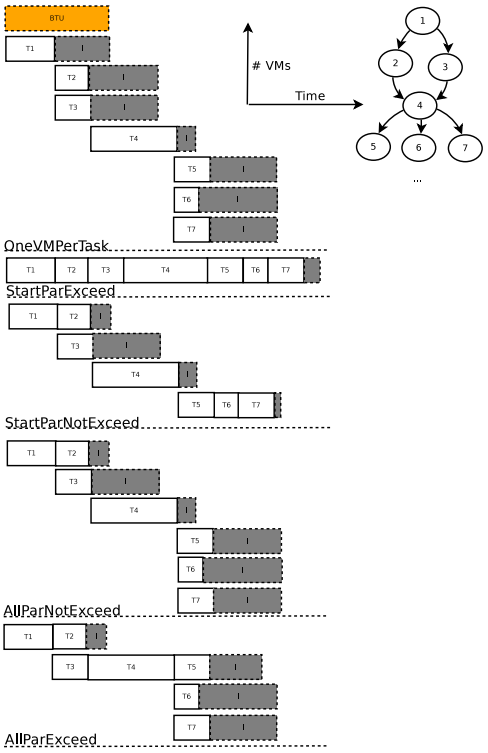


Figure 1. VM Provisioning policies examples

based on execution and transfer times is computed. Tasks are then ordered descending by their ranks and scheduled.

Level based scheduling is a similar ranking strategy that orders tasks based on their level in the workflow [12]. Each level is made up of parallel tasks. The difference from HEFT is that it allows greater flexibility in deciding the order of scheduling inside a level. This can lead to improvements where parallelism is reduced [12]. It can also be combined with CPA [1].

Cluster based scheduling comprises methods that cluster tasks located on the same path in order to reduce communication costs. Examples include PCH and HCOC.

In this work we consider seven allocation policies, namely *HEFT*, *CPA-Eager*, *Gain*, *AllParNotExceed*, *AllParExceed*, *AllPar1LnS* and *AllPar1LnSDyn*.

HEFT can be used with any of the following provisioning methods: *OneVMperTask*, *StartParNotExceed* and *StartParExceed* since they do not require any knowledge on task parallelism and can thus be used by HEFT's rank based ordering. Figure 1 exemplifies the different results HEFT produces when various provisioning strategies are used. The intent is to determine which provisioning works best with this algorithm given a range of scenarios.

AllParNotExceed and *AllParExceed* are similar SAs proposed by us that split the workflow in levels based on task parallelism. Then each task in a level is scheduled arbitrarily based on the provisioning method with the same

name. Figure 1 shows how they produce different idle times and rent costs. Their intent is to show how a simple constraint introduced by *AllParNotExceed* as compared to *AllParExceed* can influence the schedule.

The next algorithms are dynamic algorithms that rely on improving makespan and cost based on various criteria.

CPA-Eager and *Gain* rely on the *OneVMperTask* provisioning method during the initial schedule. Based on it they will attempt to increase the speed of certain VMs according to their policies. *CPA-Eager* will attempt to systematically increase the speed of VMs allocated to tasks lying on the critical path. The reason is that usually makespan is dictated by the sum of execution times of the tasks on that path. Based on the maximum budget and on the available VM types, makespan reduction is possible. *Gain* method is based on reducing the execution time of the task which gives the best speed/cost improvement when a faster VM is deployed. For this, the algorithm will compute a *gain* matrix where rows are tasks and columns VM types. Each element is computed as follows: $gain_{ij} = (execution\_time_{current} - execution\_time_{new})/(cost_{new} - cost_{current})$ [10], [12]. The task i with the greatest gain is picked and its VM is upgraded to the one that provided the maximum gain.

AllPar1LnS tries to decrease task parallelism by sequentializing multiple short tasks whose total lengths are about the same as longest tasks. Each set of sequential short tasks is mapped onto a single VM, while the long tasks are still scheduled in parallel to different VMs. The reduction is performed only after tasks are scheduled using the *AllParNotExceed* provisioning method and ranked, inside each level, by execution time. A more complex approach has been proposed in [12].

AllPar1LnSDyn is an extension to *AllPar1LnS* policy in the way that it tries to decrease the makespan inside each level by attempting to increase the VM speed inside a budget. The algorithm proceeds as follows:

First the parallelism is reduced as in *AllPar1LnS*. Then the budget for the level is computed based on the value given by a provisioning method using *AllParNotExceed*. This allows us to set up a worse case budget since in this provisioning scheme every parallel task needs to be placed o a different VM hence maximizing the rent cost. The algorithm will attempt to reduce the execution time of the longest task – which is always scheduled separately – by assigning the next fastest VM. If succeeded it checks to see whether the makespan is still determined by that task or it shifted to another VM. If the former case is true it continues to increase the speed of the VM inside the budget until the makespan shifts to other VM or there exists not a faster VM. In case the makespan shifts to another VM, the SA tries to reduce it below the execution time of the longest task by increasing – inside the budget – the speed of the VM. If this fails either due to exceeding the budget or because the makespan is still larger, the increase in speed is rolled back to the last valid configuration, i.e., one in which the budget is not exceeded and the level makespan is dictated by the longest task.

Table I depicts the relationship and characteristics of each provisioning and allocation policy.

It can be noticed that *Gain*, *CPA-Eager* and *OneVMperTask* are aimed at reducing makespan, *AllPar1LnSDyn* and *AllPar1LnS* target both cost and makespan, while *StartParExceed* minimizes cost only.

IV. Experiment Setup

Our experiments aimed at observing the relative behavior of different VM provisioning strategies for both homogeneous and heterogeneous environments. For this we tested for the homogeneous case three provisioning methods with *HEFT* and two as stand alone strategies (cf. Sect. III-A). For the heterogeneous case we tested *CPA-Eager*, *Gain*, *AllPar1LnS* and *AllPar1LnSDyn* (cf. Sect. III-B). We were particularly interested in: (1) the makespan gain/savings ratio when different provisioning strategies are used; and (2) the total idle time of the system which could be used to co-rent resources in a similar manner with what Amazon does with its spot instances.

A. Platform

Table II
Amazon EC2 prices on October 31st 2012

region	small	medium	large	xlarge	transfer out
US East Virginia	0.08	0.16	0.32	0.64	0.12
US West Oregon	0.08	0.16	0.32	0.64	0.12
US West California	0.09	0.18	0.36	0.72	0.12
EU Dublin	0.085	0.17	0.34	0.68	0.12
Asia Singapore	0.085	0.17	0.34	0.68	0.19
Asia Tokio	0.092	0.184	0.368	0.736	0.201
SA Sao Paolo	0.115	0.230	0.460	0.920	0.25

The underlying cloud model was considered to be Amazon EC2 together with its seven regions. The prices are listed in Table II and were used by Amazon for on demand instances, with one $BTU = 3,600s$. Communication costs are per GB and were considered only when moving data outside a region. They are applied if the transfer size is between $\in (1GB, 10TB]$ per month.

The small, medium, large and xlarge instances were considered to have one, two, four and eight cores each producing a speed-up of 1, 1.6, 2.1 and 2.7 times the default case with one core. The speed-ups are similar to those reported by the statistical package Stata/MP [3]. One CPU unit is roughly the equivalent of a 1.0-1.2 GHz 2007 Opteron system[4]. For the communication speed we considered small and medium to have 1Gb links while the others to have 10Gb links.

[3]http://www.stata.com/statamp/ (accessed Dec 9th 2012)

[4]http://aws.amazon.com/ec2/ (accessed Dec 7th 2012)

Table I
PROVISIONING AND ALLOCATION POLICIES

Provisioning	Task ordering	Allocation	Parallelism reduction
OneVMperTask	priority ranking	HEFT, CPA-Eager, GAIN	no
StartParNotExceed	priority ranking	HEFT	no
StartParExceed	priority ranking	HEFT	no
AllParNotExceed	level ranking + ET descending	AllPar1LnS	yes
AllParNotExceed	level ranking + ET descending	AllPar1LnSDyn	yes

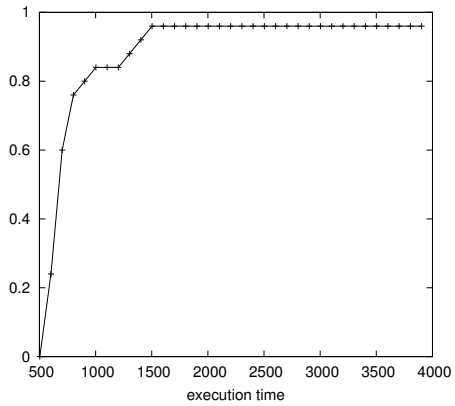


Figure 3. CDF for the Pareto distribution of execution times

The maximum allowed cost for Gain and CPA-Eager was set to for times respectively twice the cost needed in case of using HEFT with *OneVMperTask* for small instances.

The platform was simulated on a custom made simulator where the default execution time is considered to be the one for the small instance and the speed-ups are as previously mentioned. Transfer times are computed based on a store and forward model in which transfer time is equal to $size/bandwidth + latency$. Although this simplified model does not take into consideration factors such as bandwidth sharing, it suffices to get an approximate of the time needed to transfer tasks from one region to another.

Boot times for Amazon EC2's VMs are constant no matter the number of VMs and are usually less than two minutes as indicated in [22]. In our scenario we ignored them since we employ a static scheduling and can therefore use a pre-booting strategy.

B. Workflows

Three scenarios for execution times were of interest. The first one follows an analytical model of runtimes and is based on Feitelson's results [23]: a Pareto distribution having a shape of $\alpha = 2$ for execution times and $\alpha = 1.3$ for task sizes. The scale parameters were fixed to 500. Figure 3 shows the cumulative distribution function of the resulted dataset.

The second one assumes a best case scenario on which all tasks are equal in length and could fit into a single VM. In this case, given n tasks and e the individual runtime such that $ne \leq BTU$ we have for a sequential provisioning a total number of VMs equal to $m = 1$ (cost=1 BTU) while for a parallel one we have $m = n$ (cost=n BTU).

The third case assumes a worst case scenario where tasks have equal length and execution times greater than one BTU. We make it exceed one BTU even for the most powerful VM instance type, i.e., $BTU < e/2.7 < e$. In this case for the sequential provisioning we get $\lceil ne/BTU \rceil$ VMs (cost=$\lceil ne/BTU \rceil BTU$) while for the parallel provisioning $n\lceil e/BTU \rceil$ VMs (cost=$n\lceil e/BTU \rceil BTU$).

As seen in the results, for the best case we have *StartParNotExceed=StartParExceed* and *AllParNotExceed=AllParExceed*, while for the worst case *StartParNotExceed=AllParNotExceed=OneVMperTask*.

The last two cases provide us with boundaries for the fluctuation of the gain and profit.

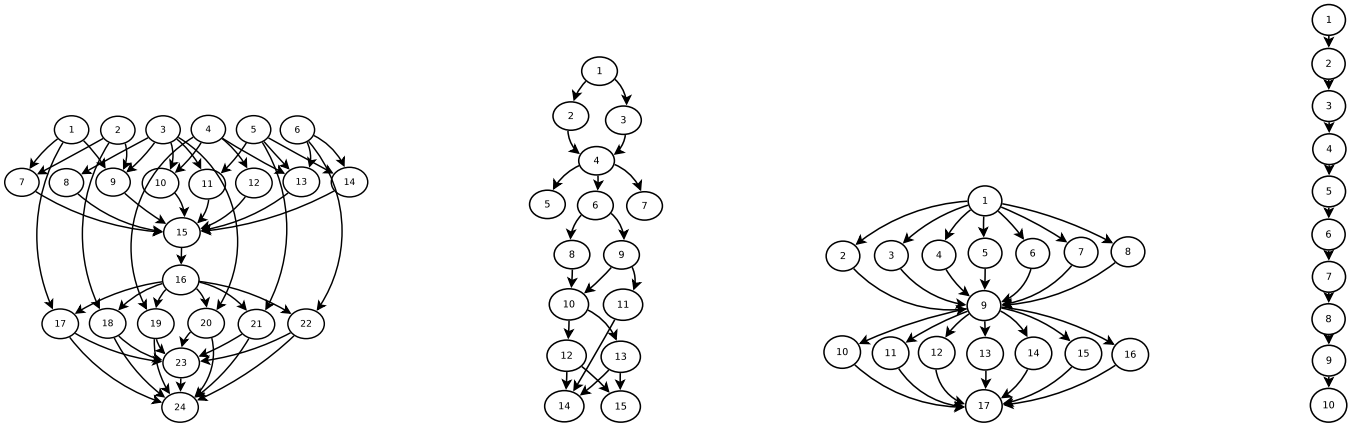
Four different workflows have been used in our tests: Montage [24], CSTEM [25], MapReduce [5] and a simple sequential one. Figure 2 depicts their structure.

Montage (cf. Fig. 2(a)) is a workflow used in astronomical image processing. Its size varying depending on the dimension of the studied sky region. In our tests we used a version with 24 tasks. The workflow dependencies are quite intermingled and data dependencies are not only from one level. This type of workflow makes it particular interesting for locality aware data intensive scheduling. At the same type the large number of parallel tasks makes it good for studying the efficiency of parallel VM provisioning policies.

MapReduce is a model first implemented in Hadoop which aims at processing large amount of data by splitting it into smaller parallel tasks. In Figure 2(c) we depict a version in which there are two sequential map phases. Its original use and the large (variable) amount of parallel tasks it can hold makes it useful in the study of various parallel VM provisioning policies for data intensive tasks and to show their benefits.

CSTEM (cf. Fig. 2(b)) is a workflow used in CPU intensive applications. Its relative sequential nature with few parallel tasks and several final tasks makes it a good candidate for studying the limits of the parallel VM provisioning policies against the *OneVMperTask* and a particular case of *StartParExceed* in which all tasks of a workflow with a single initial tasks are scheduled on the same VM.

[5]http://hadoop.apache.org/ (accessed Dec 9th 2012)



(a) Montage (b) CSTEM (c) MapReduce (d) Sequential

Figure 2. Used workflows

The sequential workflow (cf. Fig. 2(d)) is a typical example of a serial application with dependencies, e.g., makefiles. It is the opposite of the parallel intensive MapReduce model. In our paper it is used to show the limits of the parallel provisioning policies and the benefits of using *OneVMperTask* and *StartPar[Not]Exceed*.

V. Experiment Results

In this section we present the results of our experiments for computational and data intensive tasks. The *-s*, *-m* and *-l* endings used for some provisioning methods indicate the type of EC2 instance used in that case, i.e., small, medium or large.

We were mainly interested in the **makespan/savings ratio** and the **idle time** as essential factors in our study. Figures 4 and 5 depict the results.

Regarding **makespan/savings ratio** we were looking for finding the strategies that produce makespan gain or savings with respect to a default one set to *OneVMperTask* for the small instance type – marked as a filled square (cf. Figure 4) in the upper-left corner of the square that defines the area where both savings and gain occur. We also searched for any strategy which might provide stable results in terms of cost and makespan throughout the tests.

In Table III the main SAs which fall in this category are classified based on their gain/savings ratio. The first column shows algorithms that provide greater savings but less gain, the second column shows the algorithms that provide less savings but greater gain and the last column indicates the algorithms that provide a balanced gain/savings ratio.

Based on whether savings or gain is the target the following can be noticed:

Savings: Most of the SAs fall in this category. This can indicate that given the cost of a VM and the associated speed-up global cost is easier to reduce than makespan. The most interesting case is that of *AllPar[Not]Exceed* which produces a stable gain throughout the three cases but the savings fluctuate drastically as noticed in Table IV (in parenthesis there is the loss inflicted in the case of the Pareto distribution). Using small instances is the only case in which savings are positive (negative loss) while the rest can induce losses depending on the workflow type of up to 40% for the medium respectively 166% for the large instance. While for the medium instance the stable gain of 37% can be a suitable trade-off, the small gain of only 52% noticed for the large instance does not cover the loss which is about three times larger. Thus this provisioning strategy is best suited for small instances while for medium ones only when a trade-off in savings is preferred in order to achieve a better gain.

Furthermore, except the worst case with the sequential workflow, the costs inflicted by the previous two SAs can be further reduced with the *AllPar1LnS* and *AllPar1LnSDyn* algorithms.

Gain: The dynamic SAs seem to be at disadvantage in this case. In addition it seems that this situation is hard to be achieved and strongly depends on the execution times. No SA falls in this situation for the worst case while the best case has the most of them. This can indicate that if gain is the target small execution times are needed for best results. The medium and large instances dominate with the *AllParExceed-m* being a winner for Montage (best case), CSTEM (Pareto case) and MapReduce (best case). It also seems to indicate that for gain to overcome intensively parallel workflows are needed with this SA. Regarding the sequential workflow any SA that considers large instances can be used. In its case it seems that the smaller the execution times the better the results are – i.e., for the best case the gain/savings are balanced.

Table III
COMPARISON BETWEEN POLICIES THAT OFFER GAIN OR PROFIT

		$0 \leq gain\% < savings\%$	$0 \leq savings\% < gain\%$	$gain\% \approx savings\%$
Pareto	Montage	*AllParNotExceed-s* = *AllParExceed-s*, *AllPar1LnS* ≈ *StartParExceed-m*, *AllPar1LnSDyn*	–	*StartParNotExceed-m*
	CSTEM	*AllPar1LnS* = *AllPar1LnSDyn*, *StartParExceed-l*, *AllParNotExceed-s*, *AllParExceed-s*	*AllParNotExceed-m*	*StartParNotExceed-m*, *AllParExceed-m*
	MapReduce	*AllParExceed-s* = *AllparNotExceed-s*, *AllPar1LnS*	*AllParExceed-m*	*AllPar1LnSDyn*
	Sequential	**-m* except *OneVMperTask-m*, *AllPar1LnSDyn* = *AllPar1LnS* = **-s* except *OneVMperTask-s*	**-l* except *OneVMperTask-l*	–
Best case	Montage	*AllParExceed-s* = *AllParNotExceed-s*, *AllPar1LnSDyn* = *AllPar1LnS*	*StartParExceed-l* = *StartParNotExceed-l*, *AllParExceed-m* = *AllParNotExceed-m*	–
	CSTEM	*AllParExceed-m* = *AllParNotExceed-m*, *StartParExceed-l* = *StartParNotExceed-l*, *AllParExceed-s* = *AllParNotExceed-s*, *AllPar1LnSDyn* = *AllPar1LnS*	–	–
	MapReduce	*AllParExceed-s* = *AllParNotExceed-s* = *AllPar1LnSDyn* = *AllPar1LnS*	*AllParExceed-m* = *AllParNotExceed-m*	–
	Sequential	*AllPar1LnSDyn* = *AllPar1LnS* = **-s* except *OneVMperTask-s*, **-m* except *OneVMperTask-m*	–	**-l* except *OneVMperTask-l*
Worst case	Montage	*AllPar1LnS* = *AllPar1LnSDyn*	–	*StartParExceed-s* = *StartParNotExceed-s* = *AllParNotExceed-s* = 0
	CSTEM	*AllPar1LnS* = *AllPar1LnSDyn* = *AllParExceed-s*	–	*StartParNotExceed-s* = *AllParNotExceed-s* = 0
	MapReduce	*AllPar1LnS* = *AllPar1LnSDyn* = *AllParExceed-s*	–	*StartParNotExceed-s* = *AllParNotExceed-s*
	Sequential			*AllPar1LnS* = *AllPar1LnSDyn* = **-s* = 0

Table IV
SAVINGS FLUCTUATION VS. STABLE GAIN FOR ALLPAR[NOT]EXCEED

instance type	% loss interval				% max loss interval	% gain
	Montage	CSTEM	MapReduce	Sequential		
small	$[-62, 0]$ (-28)	$[-73, 0]$ (-53)	$[-58, 0]$ (-17)	$[-69, 0]$ (-70)	$[-90, 0]$	0%
medium	$[-25, 45]$ (12)	$[-46, 40]$ (-6)	$[-17, 37]$ (29)	$[-80, 33]$ (-60)	$[-80, 40]$	37%
large	$[50, 163]$ (6)	$[6, 155]$ (6)	$[-64, 134]$ (64)	$[-60, 166]$ (-20)	$[-64, 166]$	52%

Balanced: When considering a balanced gain/savings the *StartPar[Not]Exceed* SAs seem to have an advantage especially in the case of Montage and CSTEM. Depending on the nature of the execution times it can be used with small (worst case) or medium (Pareto case) instances. Their failure to provide similar results for the best case seems to indicate that these algorithms provide good results for large (and heterogeneous) execution times.

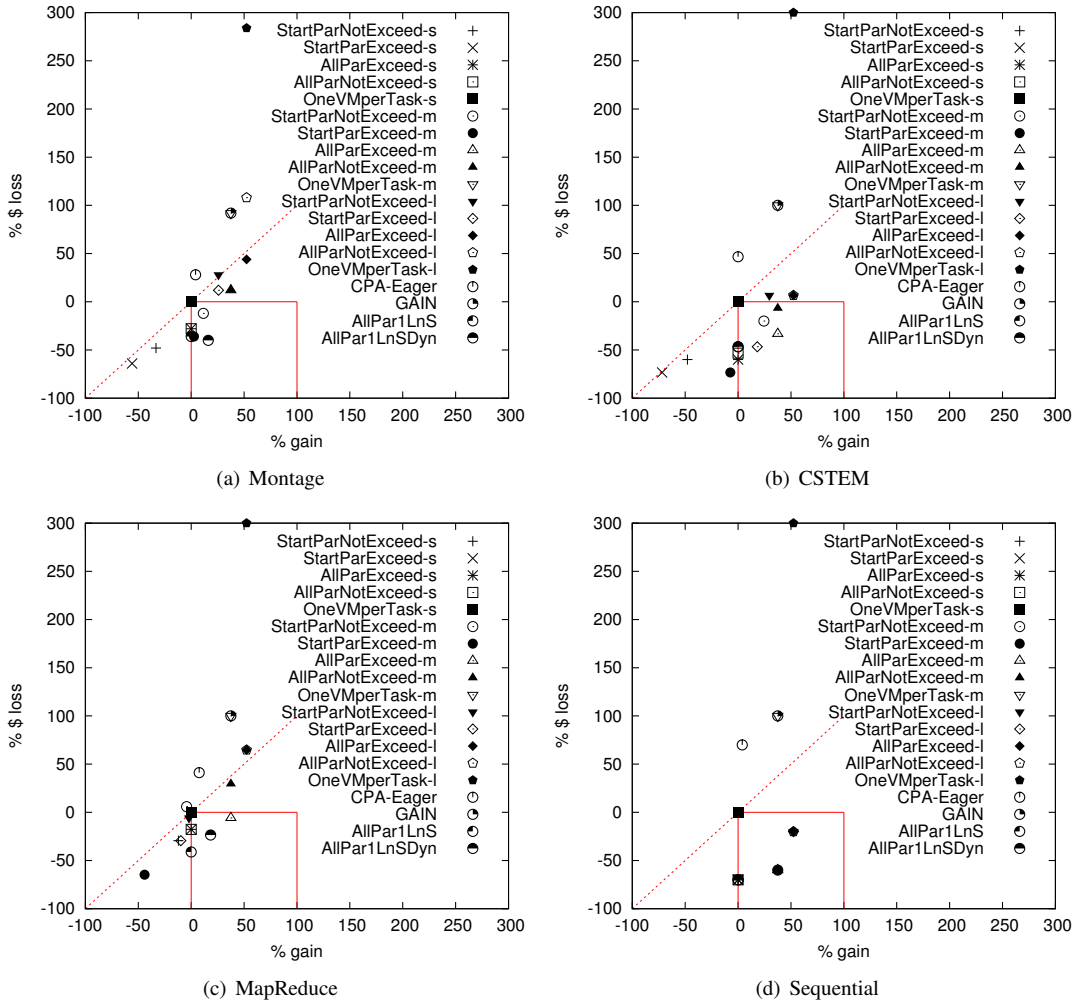
Looking at the dynamic SAs it can be stated that for *AllPar1LnSDyn* it seems the algorithms performance is proportional to the heterogeneity of the execution times. This SA is without doubt the only one that manages to remain in the target square for all cases. Nonetheless it seems that it generally produces better savings then gain thus in a gain oriented scenario other solutions should be looked for (cf. Table V). Regarding Gain and CPA-Eager, although they produce stable results throughout the three cases their profit loss ranges between $[45, 100]\%$.

While not many SAs can match the gain offered by *OneVMperTask-l* its large loss of 200-300% makes it inefficient in face of other SAs that reduce loss or even make profit.

It can be noticed that most of the algorithms present in our target square are made up of either dynamic or small/medium instances. The reason behind this is that the benefit of renting large VMs against the speed gain is of only 0.675 compared to the medium case (=0.8) and the default small one (=1). The provided speed-up is simply too small compared to the price needed to rent such VMs.

Finally for the worst case it must be noted that except *AllPar1LnSDyn* all the other algorithms from Table III do not offer improved results from the referenced *OneVMperTask-s*. For them *OneVMperTask-s* marks the upper boundary for the deterioration according to our goals.

Table V summarizes the main conclusions drawn from our experiments. Overall the dynamic *AllPar1LnSDyn* SA can be used in profit oriented scenarios, the medium instance in combination with some SAs can lead to better gain while



(a) Montage (b) CSTEM

(c) MapReduce (d) Sequential

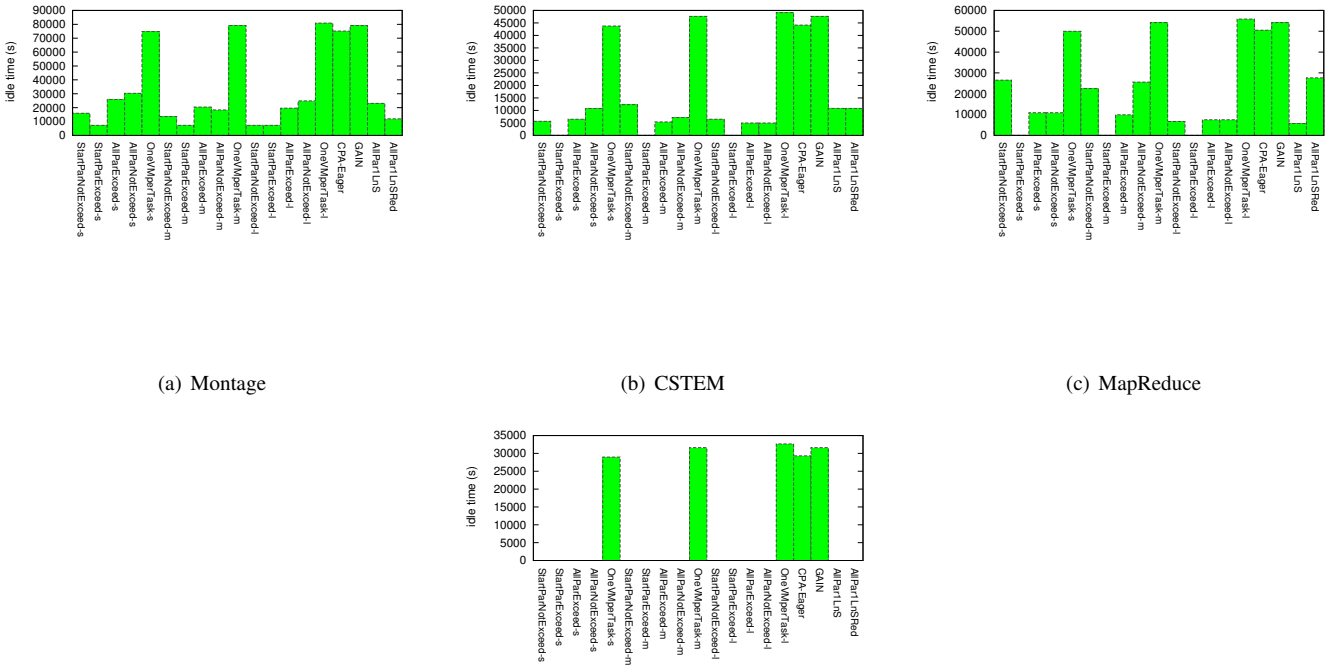
Figure 4. Makespan gain vs. cost loss for CPU intensive workflows

Table V
TEST RESULTS CONCLUSION SUMMARY

	Savings	Gain	Balance
Much parallelism (MapReduce)	*AllPar1LnSDyn*	*AllParExceed-m* ⊕ small & heterogeneous tasks	*AllParLnSDyn* ⊕ heterogeneous tasks
Much parallelism ⊕ many interdependencies (Montage)	*AllPar1LnSDyn*	*StartPar[Not]Exceed-l*, *AllPar[Not]Exceed-m* ⊕ short tasks	*StartParNotExceed-[m|s]*, ⊕ heterogeneous respectively long tasks
Some parallelism (CSTEM)	*AllPar1LnSDyn*	*AllParNotExceed-m* for heterogeneous tasks	*[Start|All]ParNotExceed-[s|m]* ⊕ long respectively heterogeneous tasks
Sequential	**-s* and *AllPar1LnSDyn* ⊕ small & heterogeneous tasks	**-l* with heterogeneous tasks	**-l* with short tasks

the large instances prove efficient for sequential workflows when gain is targeted. The results for the extreme cases (cf. Table III) provide margins for the efficiency of SAs since in these cases most of the algorithms converge to the same results.

Considering **idle time** (cf. Fig. 5) the majority of the algorithms waste between three to 13 hours, a limit which goes up to 22 total hours in case of Montage. The largest idle time are produced by the *OneVMperTask**, Gain and CPA-Eager policies. These policies also induce monetary loss and in an energy aware context their negative impact will be even more obvious since unused VMs consume energy for



(a) Montage

(b) CSTEM

(c) MapReduce

(d) Sequential

Figure 5. Idle time for CPU intensive workflows

no intended purpose. Given the large idle times their best use could be in a co-rent scenario where idle time is leased to other users and the user is partially reimbursed.

The SAs which appear to provide both gain and profit in the gain/savings scenario also produce relatively similar and smaller idle times in all cases.

In the sequential workflow scenario its serialized nature is the reason why for most methods there is no significant idle time visible.

VI. Conclusions

In this paper we have investigated the problem of workflow scheduling from the point of view of the impact VM provisioning has on the outcome. Contrary to the bag-of-task problem which has been recently studied this aspect has been left untouched. We were interested in two user objectives, that is makespan and cost. Idle time has also been studied and a relative dependency to cost has been observed.

VM provisioning and task allocation are two sides of the same process. The SA dictates the order of tasks and their mappings on VMs while provisioning specifies when and whether a new VM should be assigned or an existing one reused. As shown, due to the various structure and execution time patters a workflow can have, several provisioning schemes applied to the same SA lead to different makespans, costs or idle times. Results demonstrated a link between the SA+VM provisioning scheme, the structure of the workflow and the task execution times. Depending on what the user seeks (faster execution times, larger savings or balanced gain/savings) different combinations need to be used. They also shown that increasing the speed of all VM does not necessarily brings the best gain/cost ratio and usually large instance VMs bring gain at the detriment of considerable cost compared to small and medium ones. The exception to this rule seems to be the case of sequential workflows where powerful VMs do bring benefits. These results open the way for adaptive scheduling where the SA can be adjusted based on workflow properties and user goals.

Future work will investigate this correlation in greater detail by including custom workflows and execution times with various properties from different workloads. The aim is to determine what are the boundaries, and if the classification can be further refined, in terms on workflow structure and execution times for the results depicted in Table V.

References

[1] E. Caron, F. Desprez, A. Muresan, and F. Suter, "Budget constrained resource allocation for non-deterministic workflows on an iaas cloud," in *Algorithms and Architectures*

for Parallel Processing, ser. Lecture Notes in Computer Science, Y. Xiang, I. Stojmenovic, B. Apduhan, G. Wang, K. Nakano, and A. Zomaya, Eds. Springer Berlin Heidelberg, 2012, vol. 7439, pp. 186–201.

[2] J. Gu, J. Hu, T. Zhao, and G. Sun, "A new resource scheduling strategy based on genetic algorithm in cloud computing environment," *JCP*, pp. 42–52, 2012.

[3] J. O. Gutierrez-Garcia and K. M. Sim, "A family of heuristics for agent-based elastic cloud bag-of-tasks concurrent scheduling," *Future Generation Computer Systems*, no. 0, pp. –, 2012.

[4] E. Michon, J. Gossa, and S. Genaud, "Free elasticity and free CPU power for scientific workloads on IaaS Clouds," in *18th IEEE International Conference on Parallel and Distributed Systems*. Singapour, Singapore: IEEE, Dec. 2012.

[5] D. Villegas, A. Antoniou, S. M. Sadjadi, and A. Iosup, "An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, ser. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 612–619.

[6] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations." in *Integrated Network Management*. IEEE, 2007, pp. 119–128.

[7] E. Deelman, G. Singh, M. Livny, G. B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in *SC*, 2008, p. 50.

[8] H. Zhao and R. Sakellariou, "An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm," in *Euro-Par 2003 Parallel Processing*, ser. Lecture Notes in Computer Science, H. Kosch, L. Bszrmnyi, and H. Hellwagner, Eds. Springer Berlin Heidelberg, 2003, vol. 2790, pp. 189–194.

[9] A. Radulescu and A. van Gemund, "A low-cost approach towards mixed task and data parallel scheduling," in *Parallel Processing, International Conference on, 2001.*, sept. 2001, pp. 69 –76.

[10] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, "Scheduling workflows with budget constraints," in *in Integrated Research in Grid Computing, S. Gorlatch and M. Danelutto, Eds.: CoreGrid series*. Springer-Verlag, 2007.

[11] C. Lin and S. Lu, "Scheduling scientific workflows elastically for cloud computing," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, july 2011, pp. 746 –747.

[12] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 49:1–49:12.

[13] K. Le, R. Bianchini, J. Zhang, Y. Jaluria, J. Meng, and T. D. Nguyen, "Reducing electricity cost through virtual machine placement in high performance computing clouds," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 22:1–22:12.

[14] K. Liu, "Scheduling algorithms for instance-intensive cloud workflows," Ph.D. dissertation, University of Swinburne Australia, 2009.

[15] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, "Cost optimized provisioning of elastic resources for application workflows," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1011 – 1026, 2011.

[16] H. Mohammadi Fard, R. Prodan, and T. Fahringer, "A truthful dynamic workflow scheduling mechanism for commercial multi-cloud environments," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2012.

[17] L. Bittencourt and E. Madeira, "Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds," *Journal of Internet Services and Applications*, vol. 2, pp. 207–227, 2011, 10.1007/s13174-011-0032-0.

[18] L. F. Bittencourt and E. R. M. Madeira, "A performance-oriented adaptive scheduler for dependent tasks on grids," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 9, pp. 1029–1049, Jun. 2008.

[19] S. Pandey, L. Wu, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, april 2010, pp. 400 –407.

[20] 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*, pp. 1–38, 10.1007/s11227-011-0578-4.

[21] E. Hwang and K. H. Kim, "Minimizing cost of virtual machines for deadline-constrained mapreduce applications in the cloud," in *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, sept. 2012, pp. 130 –138.

[22] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *IEEE CLOUD*, 2012, pp. 423–430.

[23] "Workload modeling for computer systems performance," (accessed Oct 12th 2012). [Online]. Available: http://www.cs.huji.ac.il/ feit/wlmod/

[24] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, no. 3, pp. 219–237, Jul. 2005.

[25] A. Doğan and F. Özgüner, "Biobjective scheduling algorithms for execution time-reliability trade-off in heterogeneous computing systems*," *Comput. J.*, vol. 48, no. 3, pp. 300–314, May 2005.