2011 Third IEEE International Conference on Coud Computing Technology and Science

Cost-Efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds

Ruben Van den Bossche
Department of Mathematics
and Computer Sciences
Universiteit Antwerpen
Antwerp, Belgium
Email: Ruben.VandenBossche@ua.ac.be

Kurt Vanmechelen
Department of Mathematics
and Computer Sciences
Universiteit Antwerpen
Antwerp, Belgium

Jan Broeckhove
Department of Mathematics
and Computer Sciences
Universiteit Antwerpen
Antwerp, Belgium

***Abstract*—Cloud computing offerings are maturing steadily, and their use has found acceptance in both industry and research. Cloud servers are used more and more instead of, or in addition to, local compute and storage infrastructure. Deciding which workloads to outsource to what cloud provider in such a setting, however, is far from trivial. This decision should maximize the utilization of the internal infrastructure and minimize the cost of running the outsourced tasks in the cloud, while taking into account the applications' quality of service constraints. Such decisions are generally hard to take by hand, because there are many cost factors, pricing models and cloud provider offerings to consider. In this work, we tackle this problem by proposing a set of heuristics to cost-efficiently schedule deadline-constrained computational applications on both public cloud providers and private infrastructure. Our heuristics take into account both computational and data transfer costs as well as estimated data transfer times. We evaluate to which extent the different cost factors and workload characteristics influence the cost savings realized by the heuristics and analyze the sensitivity of our results to the accuracy of task runtime estimates.**

I. Introduction

The IT outsourcing model in which access to scalable IT services hosted by external providers can flexibly be acquired and released in a pay-as-you-go manner, has seen an increasing adoption recently under the term *cloud computing*. A key differentiator of the cloud computing model compared to other outsourcing models such as managed hosting is that it allows for more automation and flexibility in the acquisition and release of IT resources, while minimizing the amount of direct interactions required between service provider and consumer.

The model's recency and the hype surrounding cloud computing has up until now precluded the broad acceptance of a single definition of what cloud computing exactly is, introducing ambiguity [1]. An often used classification categorizes cloud services as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS), depending on the service's focus on delivering respectively IT infrastructure, software development frameworks or a finished software product [2].

Our work focuses on the IaaS service class in which a consumer can acquire access to compute, storage, and networking capacity at the provider's site. To acquire compute resources, a consumer launches an *instance* on the cloud provider's infrastructure, thereby specifying the instance's characteristics such as the available processing power, main memory and I/O capacity. Most commonly, the notion of an instance materializes into a virtual machine that is launched on the provider's physical IT infrastructure. Virtualization technology enables the provider to increase infrastructure utilization by multiplexing the execution of multiple instances on a single physical server, and allows one to flexibly tune the individual characteristics of an instance. Nevertheless, most providers predefine combinations of instance characteristics in a fixed number of *instance types*.

In the last few years, the IaaS market has quickly matured, with providers rapidly diversifying their product offerings in terms of pricing plans, instance types and services. Amazon's Elastic Compute Cloud (EC2) offering for example, has increased the number of instance types from one to eleven in less than four years, and moved from a single on-demand pricing model to three different pricing models with the addition of *reserved instances* and dynamically priced *spot instances*. Likewise, it has expanded the number of geographical *regions* in which instances can be launched from one to five. The choice of the instance's region both influences the network characteristics in terms of available bandwidth and latency, as well as the cost of running the instance. Finally, the number of IaaS providers has increased significantly as well [3], with each provider differentiating itself in terms of services offered, prices charged for different resources (ingress and egress network bandwidth, memory, CPU) and performance.

Consequently, consumers are faced with an increasing complexity in making cost-optimal decisions when matching their infrastructure requirements with the IaaS services currently available. This issue is exacerbated by the fact that consumers often also own internal IT infrastructure. Through the creation of hybrid clouds [4], [5], one can use this internal infrastructure in tandem with public cloud resources, thereby capitalizing on investments made, and catering for specific application requirements in terms of data confidentiality, security, performance and latency.

In this paper, we focus on the optimization problem of allocating resources from both a private cloud and multiple public cloud providers in a cost-optimal manner, with support

978-0-7695-4622-3/11 $26.00 © 2011 IEEE
DOI 10.1109/CloudCom.2011.50

for application-level quality of service constraints such as minimal throughput or completion deadlines for the application's execution. We analyze how this optimization problem can be tackled in the context of resource provisioning for batch workloads through the use of different scheduling heuristics, and investigate the impact of workload model parameters on the cost reductions that can be achieved. Our workload model considers non-preemptible and non-migratable workloads with a hard deadline that are characterized by CPU, memory and data transmission requirements. To our knowledge, we are the first to address the resource allocation problem in hybrid clouds in this form. In previous work [6], we presented a linear programming formulation of the optimization problem for a static set of applications. We experienced large solve time variances, undermining the feasibility of this approach for more complex and large-scale scenarios. In this contribution, we tackle this issue by developing custom heuristics for cost-efficient scheduling. We also extend our workload model to include applications that are associated with a data set, which allows us to take *data transmission speeds* and *data locality* into account during the scheduling process. Finally we further generalize our approach to support online arrival of applications.

II. Problem Domain

In a hybrid cloud setting, the execution of applications occurs through the deployment of virtual machine instances on resources internal to the consumer organization, as well as resources provided by public cloud providers. The characteristics of the virtual hardware on which these instances are executed are determined by the –mostly provider-specific– definition of *instance types*. An instance type fixes the number and speed of CPUs, the amount of memory, local storage and the architecture of the virtual machine instance. Note that some providers (e.g. CloudSigma) don't have fixed instance types, and allow users to customize and fine tune the cloud servers to fulfill the exact hardware needs of their applications. Selecting the best cloud server configuration for an application in a continuous spectrum of hardware properties is a task that ultimately results in one or a few server configurations with an associated cost suitable for the application. From the perspective of a scheduler, these cloud configurations are similar to the provider-specific instance types. Although simple, adding support for fine grained user-configured instance types would thus hardly affect the results in this work. Therefore, we assume in this study that every public cloud provider offers a number of fixed instance types and associates a price with the execution of a virtual machine on each instance type. A cloud provider charges for the time the instance has run, and commonly handles a discrete *billing interval* that has to be fully paid for. The cloud providers modeled in this contribution –Amazon EC2, Rackspace and GoGrid– express and charge prices on a hourly basis. This implies that running a cloud server for 61 minutes costs the same as running the cloud server for two hours. A provider also charges for each gigabyte of network traffic used by the consumer's applications. In this regard, a distinction is made between the price for inbound and outbound traffic. For now, we do not consider the costs for additional services such as persistent data storage, monitoring or automated workload balancing.

For the purposes of this study, the consumer's internal infrastructure is assumed to support a resource allocation model that is similar to that of a public cloud provider in that it also allows for the flexible deployment of virtual machines. Such a system is commonly referred to as a *private cloud*.

Heterogeneous workload characteristics and divergent user requirements complicate the problem of identifying the best cloud provider and instance type for a specific application. In this paper, we narrow our view to only some of the characteristics and requirements, of which we believe they constitute a good starting point to demonstrate the operation and implementation of a hybrid cloud scheduler.

Our current application model focuses on batch type workloads. Examples of workloads that fit this model are simulation experiments that consist of a bag of independent instances of a simulation, image and video rendering codes and highly parallel data analysis codes. Common types that are not covered by our model are coupled computations that are distributed over multiple systems, workloads with a non-trivial workflow structure that includes precedence constraints, and online applications such as web or database servers.

We assume that each application's workload consists of a number of trivially parallel[1] tasks. The instance types on which tasks are executed are modelled as a set of *unrelated parallel machines* [7]. This means that every task can run on each instance type and that each instance type has a task-specific speed by which it executes a task.

In addition, an application is also associated with a data set. We assume that the full dataset has to be transferred to the cloud provider on which the application is running. For simplicity reasons, an application cannot be scheduled on more than one cloud provider. Finally, each application is associated with a hard completion deadline. Before this deadline, all computational tasks in the application must be fully executed.

Determining estimates of task runtimes and modeling speedup of tasks on multiple processors are complex problems that have been extensively researched and fall beyond the scope of this paper. Depending on the type of the application, we acknowledge that making exact predictions can be difficult and sometimes impossible. A similar remark holds for the estimated amount of network traffic that tasks generate for the communication of their input and output data. The extent to which a certain schedule is optimal is therefore dependent on the accuracy of the provided runtime estimates. In the evaluation section of our study, we present the results of a sensitivity analysis of the proposed algorithms to runtime estimation errors. For workloads that are executed repeatedly because they form a structural part of an organization's business processes, we argue that it should be possible to build

[1] Applications are called trivially parallel when the tasks that form the application are fully decoupled and have no precedence constraints.

fairly accurate models. Such a setting allows for application-level benchmarking in order to map out the effect of using a particular instance type on application speedup.

III. Workload Model

Due to the lack of sufficient real-world data, especially with regard to application data set sizes and deadline constraints, we have to resort to a partially synthetic model for generating application instances for the experiments in Section IV.

Every scheduling experiment has a duration of one week, in which new applications arrive every second following a Poisson distribution with $\lambda = 0.002$. This results in an average interarrival time of 2000 seconds, or about 1200 applications in a week. With these parameters, we aim to generate enough load to tax the private cloud beyond its saturation point so that it becomes advantageous to use a hybrid cloud setup.

The number of tasks per application is uniformely distributed between 1 and 100. Task runtimes within one application are typically not independent of each other. Therefore, we assign an application a *base runtime*. Following [8], the application's base runtime is modeled as a Weibull distribution. The distribution's parameters are derived from the Parallel Workloads Archive's *SDSC IBM SP2* workload trace[2]. The runtime of each individual task is then drawn from a normal distribution, with the application's base runtime as the mean μ and a relative *task runtime standard deviation* σ. A task runtime standard deviation of 0% will lead to identical task runtimes, equal to the application's base runtime, while for example $\sigma = 100\%$ results in high runtime variations. Unless mentioned otherwise, the relative task runtime standard deviation was fixed at 50%.

Tasks can typically run on more than one instance type. Running a task on an instance type with for example 1 CPU and 1.7 GB memory will probably –but not always– be slower than running the same task on an instance type with 4 CPUs and 7.5 GB memory. Modeling speedup of parallel tasks on multiple processors without thorough knowledge on the operation of the application is a delicate and complex task. We model the speedup after Amdahl's law, which is used to find the maximum expected improvement of a system when only a part of it can be improved. We assume that a task has a sequential fraction with length c_1 and a parallelizable fraction with length c_2. The execution time of the task then equates to $c_1 + c_2/n$, where n is the number of processors available on the instance type. We describe the size of the parallel component as a percentage of the total runtime. We use a uniform distribution between 0% and 100%, covering both highly sequential and highly parallelizable applications. Note that in this workload model, we do not factor in the speed of the individual cores of the instance type to determine the runtime, and assume it to be homogeneous across the different instance types[3]. Next to the number of CPU cores, the amount of memory available on an instance type IT_{mem} can also influence the runtime of a task. Applications get assigned a *minimum memory requirement* App_{MemReq}. The runtime for tasks run on instance types that meet the application's memory requirement remains unaltered. If $App_{MemReq} > IT_{mem}$, a task suffers a slowdown equal to IT_{mem}/App_{MemReq}. Although this linear slowdown is optimistic and might be worse in practice, it satisfies our need to differentiate between instance types. The minimum memory requirement is normally distributed with an average of 1.5 GB and a standard deviation of 0.5 GB.

Taking into account the application's base runtime, task runtime variance, parallel component size and memory requirement, the runtimes for each of the application's tasks on each of the available instance types can be determined by the application's owner before submission to the scheduler.

In our workload model, applications have a deadline before which all tasks in the application have to be completed. This deadline will determine to a large extent on which instance type the tasks will run. Without deadlines, all tasks would be executed on the slowest and cheapest instance type. We model the application deadline as a factor of its runtime on the *fastest instance type* available. A small deadline factor results in a tight deadline, a deadline factor < 1 will always result in unfeasible deadlines. To cover both tight and loose deadlines in our workload, the deadline factor is uniformly distributed between 3 and 20.

Finally, an application is also linked to a *data set* that is used by each of the application's tasks. The data set size of an application is normally distributed with averages ranging from 0 GB for purely computational tasks to 2500 GB for data-intensive applications with a very large data set. The relative standard deviation is fixed to 50%.

A overview of the distributions used in this paper is given in Table I.

Table I
Workload model distributions

Characteristic	Distribution
App. arrival rate (seconds)	Poisson(0.002)
Tasks per application	Uniform(1,100)
Base runtime (hours)	Weibull($\lambda = 1879$, $\beta = 0.426$)
Task runtime standard deviation (%)	50%
App. parallel fraction (%)	Uniform(0, 100)
App. memory requirement (GB)	Normal($\mu = 1.5$, $\sigma = 0.5$)
App. deadline factor	Uniform(3,20)
App. data set size (GB)	Normal(varying μ, $\sigma = 50\%$)

IV. Scheduling Components

We introduce a hybrid scheduling solution suitable for the problem domain described in Section II. The proposed solution consists of three loosely coupled components:

- The *public cloud scheduler* decides for an incoming application –with given runtimes for each of the available instance types and with a given data set size– on which of the public cloud providers to execute that application.

[2]http://www.cs.huji.ac.il/labs/parallel/workload/

[3]Generalizing the model to non-homogeneous CPU speeds however does not impact our scheduling algorithms and could be achieved by adding an additional scaling factor for the task's runtime on a given instance type.

It takes into account the cost for execution and transferring data, as well as differences in data transfer speeds between the cloud providers.

- The *private cloud scheduler* schedules an incoming application on a private cloud infrastructure, taking into account the limited amount of resources available.
- The *hybrid cloud scheduler* decides whether an incoming application can be scheduled on the organization's private infrastructure or has to be offloaded –at a cost– to the public cloud. It may consider a variety of factors, ranging from private cloud utilization or queue wait time to the public cost of the application or budget constraints.

Each of the following subsections explain how our scheduling components work, which policies they use and demonstrate their operation using one or more scenarios. All simulations were performed with a simple Java based discrete time simulator, and executed on a system with two Intel Xeon E5620 processors with 16 GB memory. The simulation of running one hybrid scenario in which 1267 applications arrive over the course of one week took on average less than 12 minutes, with outliers for bigger private cloud sizes to 35 minutes, which is still less than two seconds per application scheduled.

A. Public Cloud Scheduling

The public cloud scheduling component schedules each incoming application on a public cloud provider. We therefore introduce a cost function $Cost$ (Equation 1) for an application a, which will consist of the computational costs and the data costs of running the application on a public cloud provider p. We define the set of tasks of application a as T_a. A provider p supports the set of instance types IT_p. The cost of running a task t on provider p with instance type it is defined as $C_{task}(t, p, it)$, and is elaborated in Equation 2. The time to deadline of application a is DL_a. The runtime of task t on instance type it is given by $RT_t^{it,p}$, which is rounded up to the nearest discrete time unit (i.e. 1 hour) of provider p's billing interval. The cost of running an instance of type it on provider p for one time unit is defined as C_{it}^p. The scheduler calculates the cost for running the application on each available cloud provider p, and schedules all tasks of an application on the cheapest cloud provider.

The first cost factor in Equation 1 is the computational cost C_{comp} (Equation 3) to run each task of application a within deadline constraints on the cheapest instance type.

$$Cost = C_{comp} + C_{data} \tag{1}$$

$$C_{task}(t, p, it) = \begin{cases} RT_t^{it,p} \cdot C_{it}^p & : \quad RT_t^{it,p} \leq DL_a \\ \infty & : \quad RT_t^{it,p} > DL_a \\ \infty & : \quad it \notin IT_p \end{cases} \tag{2}$$

$$C_{comp} = \sum_{t}^{T_a} \min_{it}^{IT_p} \left(C_{task}(t, p, it) \right) \tag{3}$$

Next to the computational costs, the cost function in Equation 1 also takes the data transfer costs C_{data} (cfr. Equation 4) for the application's data set into account. The data set size of application a is defined as D_a (in GB), and the inbound and outbound network traffic prices per GB of provider p are denoted by NW_p^{in} and NW_p^{out} respectively. Further on, we only focus on inbound traffic and assume that D_a has to be fully transferred to the cloud provider on which a is running.

$$C_{data} = D_a \cdot NW_p^{in} \tag{4}$$

We introduce a public cloud landscape in which users run applications on three providers, modeled exactly after the on-demand offerings of Amazon's EC2[4], Rackspace and GoGrid, with all their prices and instance type configurations. The application's average data set size varies from 0 GB to 2.5 TB, with a relative standard deviation of 50%. We compare the difference between our cost-efficient scheduler with (*CE-Data*) and without (*CE-NoData*) taking into account the data transfer cost. The results are shown relative to a *Random* policy, in which for each arriving application a cloud provider is picked randomly. Figure 1 illustrates the impact of taking data costs into account when scheduling applications on the public cloud, and shows the influence of the data set size of an application on the differences between both policies.

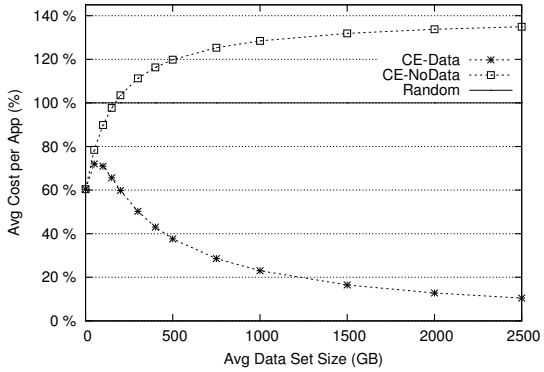


Figure 1. Impact of data costs.

Next to the expected cost reduction of our cost-efficient scheduler compared to a random policy, we also observe a significant cost reduction of the scheduling policy that takes data costs into account compared to the same policy that only takes note of the computational costs. The second term in cost equation 1 appears to be at least as important as the first.

The application's data set size does not only affect the data transfer costs to a cloud provider, but may also influence the runtimes of individual tasks. Before a task can run on a cloud provider, the application's data set should be transferred completely. If an application has a large data set, the additional time necessary to transfer the data to the cloud provider may require tasks to be scheduled on a faster instance type in order to meet the application's deadline. It is therefore important to also take the data transfer times from the consumer to cloud

[4]For Amazon EC2, prices are taken from the US-east region, and their "cluster" and "micro"-instances are disregarded.

provider into consideration. Some cloud providers, such as Amazon, provide multiple regions in different continents to satisfy the demand for *data locality*.

Differences in data transfer speed between the different regions can be significant. As an example, we measured the average bandwidth available from the University of Antwerp in Belgium to three of the Amazon EC2 regions using the iperf[5] bandwidth measurement tool. Every half hour over the course of one day, we measured the available bandwidth to the `us-east`, `us-west` and `eu-west` regions. The averages of these measurements are shown in Table II. In our case, transferring data to `eu-west` is more than twice as fast as transferring data to the cheaper `us-east` region.

Table II
AVAILABLE BANDWIDTH FROM UA TO EC2

Zone	Avg (MB/s)	Time/GB (s)	Stdev
`us-east`	15.04	68	11.91%
`us-west`	10.8	95	8.03%
`eu-west`	39.15	26	15.85%

In order for the public cloud scheduler to take data locality into account when scheduling an application, we update C_{task} in Equation 2 so that only these instance types are available for which the sum of the data transfer time and the task runtime is smaller than the time to deadline DL_a. Figure 2 demonstrates this feature. In this experiment, the Rackspace and GoGrid providers are omitted, and all applications are scheduled on one of the previously benchmarked regions of Amazon EC2. The application's average data set size ranges from 0 GB to 2.5 TB, and we use the *CE-Data* policy to calculate computational and data transfer costs. In total, 1297 applications are scheduled. The figure shows the amount of applications scheduled in each of the regions. We observe that applications without a data set are all scheduled in the `us-east` region, as it provides the cheapest compute capacity. The larger the data set, the more applications are executed in the nearby `eu-west` region in order to succesfully meet the applications' deadline constraints. The percentage of applications not meeting their deadline increases quickly as average data set sizes exceed 400 GB. This can be explained by the fact that in our workload model, the application's deadline is determined as a factor of the fastest possible task runtime, without considering data transfer times. Increasing the average data set size in an experiment consequently results in tighter deadlines and can lead to the sum of the data transfer time and the task runtime on the fastest instance type to exceed the deadline.

B. Private Cloud Scheduling

Our private infrastructure is modeled as a simple queuing system in which incoming requests for an application's tasks are handled in a first-come first-served manner without backfilling. The private infrastructure obviously does not have an

[5]http://iperf.sourceforge.net/

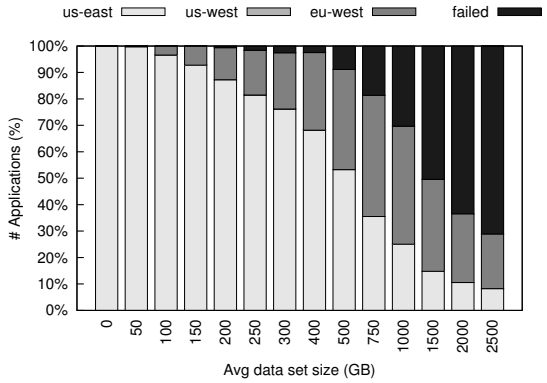


Figure 2. Impact of data locality.

"infinite" capacity. The number of tasks running concurrently on our private cloud is thus limited: the sum of the number of CPUs of all running instances may not exceed the total number of CPUs available. Overhead for virtualization as well as unusable leftovers due to the partitioning of physical servers into multiple instances are not considered. The private cloud supports four instance types, with respectively 4, 8, 16 and 26 CPU's and 4 GB, 8 GB, 16 GB and 26 GB memory. A task is scheduled on the instance type on which it induces the smallest *load*, which is defined as the runtime multiplied by the amount of CPUs of the instance type. The instance type for a task is selected at the moment the task arrives at the private scheduling component, and won't change afterwards.

This scheduling strategy does not provide an optimal schedule with respect to maximal adherence to application deadlines, cluster utilization or generated user value [9]. It does however provides a simple scheduling heuristic that allows for the exact calculation of the *queue wait time* of a task. This permits a hybrid scheduler only to schedule applications on the private cloud infrastructure that with certainty finish before deadline. This enables us to evaluate the consequences of the hybrid cloud scheduling decisions and make abstraction of the private scheduler's performance. The approach of using other metrics such as the private cluster's utilization, the number of applications in the queue or using an integrated hybrid advance reservation system are left for future work.

From the perspective of the hybrid scheduler, running an application on the private cloud is free of cost, and the applications' data sets are available free of charge and without a transfer delay. This will cause a cost-efficient hybrid scheduler to maximize the local infrastructure's utilization, and offload applications only when absolutely necessary. In an organization with an internal billing mechanism, it is however possible that users will be charged for their use of the private compute infrastructure. The development of a fair internal billing system for organizations with a hybrid cloud setup that takes into account public and private operational and capital expenses is beyond the scope of this contribution.

C. Hybrid Cloud Scheduling

In a hybrid setup, additional logic is required to decide which applications to run on the organization's own computing infrastructure and which ones to offload to a public cloud provider. After making this decision, the hybrid scheduler hands over control to the public or private scheduling component. In all of the experiments in this subsection, we use a public scheduling component with the *CE-Data* policy.

The hybrid scheduling component has to decide on the allocation of application workloads to the public and private cloud components. It should provide an efficient solution with respect to the total cost for running all applications, while minimizing the number of applications missing their deadlines due to queue congestion or high data transfer times. In the remainder of this section, we propose a hybrid scheduling mechanism that takes into account the *potential cost* of an application when scheduled on a public cloud provider. We compare this *Cost Oriented* approach with a first-come first-served *Private First* scheduler.

Such a simple *Private First* scheduler adds all incoming applications to the queue of the private cloud, unless the *queue wait time* for that application indicates that meeting the application's deadline is impossible by the sole use of the private cloud. In that case, the application is scheduled on the cheapest public cloud provider. Because the runtimes of all tasks in the queue are known, calculating the expected wait time in the queue for a set of tasks is trivial.

This FCFS scheduling methodology can result in a cost-inefficient schedule when, for example, application B with a cost of \$100 on the public cloud arrives shortly after application A with a potential cost of \$50. It is then possible that application B has to be executed externally to meet its deadline constraints, resulting in an additional cost for the system of \$50. Our *Cost Oriented* approach tries to avoid these errors by calculating on arrival the cost of running the application on the public cloud within deadline, and giving priority to the most expensive applications on the private cloud. This *potential cost* can be calculated using the public cloud scheduling algorithms described in Subsection IV-A. In this approach, applications can't be scheduled on arrival because that would again result in handling them in a first-come first-serve manner. On the other hand, delaying the scheduling decision on the hybrid scheduling level is delicate because it may result in a higher cost due to the need for more expensive instance types to complete the application within deadline. The technique of waiting until after the point where the application cost increases (and there is need for a more expensive instance type) could theoretically still result in cost reductions. In practice this turned out to happen only by exception and never outweighed the additional costs brought about by badly made decisions. The hybrid scheduler therefore uses the following strategy :

1) Determine max_{cost}: the time up to which the application's cost of execution remains constant and no instance type switch is required.
2) Determine max_{DL}: the time up to which adhering to the application's deadline remains feasible.
3) Delay the scheduling decision up to $min(max_{cost}, max_{DL})$.

Before scheduling, applications are sorted on their *potential cost*, normalized to the application's *load*[6] on the private cloud. This way, the most expensive applications are treated first. The full scheduling logic for which the pseudocode is shown in Algorithm 1, is executed every N seconds. A small value of N allows the scheduler to fine-tune every scheduling decision with a high granularity, while an increased value of N may involve a smaller computational overhead. We found a value for N corresponding to 10 minutes to be suitable in our simulations.

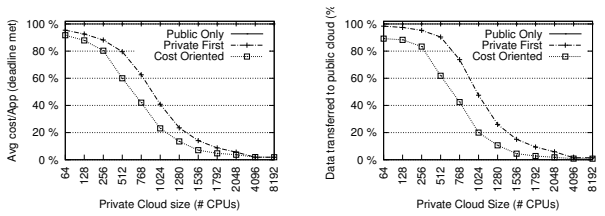
```
Input: A = {Unscheduled applications}
SortMostExpensiveFirst(A)
foreach Application a ∈ A do
    if EstimatedQueueWaitTime(a)
       + EstimatedPrivateRuntime(a) > DL_a then
        // Deadline cannot be met on private
        ScheduleOnPublic(a);
    else if EstimatedQueueWaitTime(a)
       + EstimatedPrivateRuntime(a) + N > DL_a then
        // Deadline will become unfeasible
        ScheduleOnPrivate(a);
    else if TimeToPriceIncrease(a) - N < 0 then
        // Application will become more expensive
        ScheduleOnPrivate(a);
    else
        // Let a wait until next round
    end
end
```

Algorithm 1: *Cost Oriented Scheduling* Pseudocode

We demonstrate the operation of the hybrid scheduling components by adding a private cloud to the experimental setting with the different Amazon EC2 regions used in Section IV-A. Computational costs, data transfer costs as well as data transfer times are taken into account. The average data set size in this experiment is 50 GB with a relative standard deviation of 50%. We compare the *Cost Oriented* scheduling policy to the *Private First* policy as well as a *Public Only* scheduler, which schedules all applications on the public cloud, and a *Private Only* scheduler, which adds all applications to the private queue. In Figure 3(a), the average cost per application executed within deadline is shown for the *Cost Oriented* and *Private First* policies, relative to *Public Only* policy. It is to be expected that adding a "free" private cluster results in significant cost reductions, but we also observe that our *Cost Oriented* logic is up to 50% less expensive than the *Private First* policy. This cost reduction can be achieved by sending significantly less data-intensive applications to the public cloud. This is illustrated in Figure 3(b), which displays the total amount of data transferred to the public cloud, relative to the total amount of data of all applications.

In all algorithms used in this paper, we assume that the exact runtimes are known and provided by the consumer a

[6]The load of a is defined as the sum over all tasks of the runtime multiplied by the amount of CPUs of the instance type.



(a) Cost (b) Outgoing data

Figure 3. Impact of hybrid scheduling policies on *Cost* and *Outgoing data*.

priori. Our *Cost Oriented* algorithm, and to a lesser extent also the *Private First* algorithm, depend on the accuracy of the provided runtimes as they both use the exact queue wait time, and the *Cost Oriented* policy attempts to delay the scheduling decision as much as possible up until the application's deadline. Because user provided runtimes will never be exact, we evaluate the degree of sensitivity of the algorithms to runtime estimation errors. Therefore, scheduling algorithms now base all their decisions on *estimated runtimes* instead of exact runtimes. The average error on these runtimes ranges from 0% to 50%. In this experiment, the three Amazon EC2 regions are used as public cloud together with a private cluster of 512 CPUs. Applications have an average data set size of 50 GB. The impact of these runtime estimation errors on the number of applications meeting their deadlines and on the average cost per application is illustrated in Figure 4(a) and Figure 5(a) respectively. We can observe a dramatic decrease in the number of deadlines met for the *Cost Oriented* approach. This is accompanied by an increase in cost per application, which is the result of the high cost that has to be paid for partially executed applications with unexpected failed deadlines.

These results constitute an impediment to the use of the proposed algorithms in a setting wherein accurate runtime estimates are difficult to attain. A countermeasure to reduce the number of failed deadlines can be to increase the conservativeness of the scheduler by deliberately overestimating the given runtimes with a certain percentage. In the following three experiments, we add an *Overestimation factor* of 10%, 50% and 100% to the estimated runtime of every task. All other experiment settings remain unchanged. The results with regard to the number of deadlines met and the average cost are shown in Figures 4 and 5. Overestimating the user provided runtimes also slightly influences the cost for scheduling an application in a public cloud environment, as the public scheduling component may have to pick a more expensive instance type. Compared to Figure 5(a), the absolute cost for *Public Only* in Figures 5(b)–5(d) increases less than 2% for an *overestimation factor* of 10%, less than 9.1% for an *overestimation factor* of 50% and less than 12.2% for an *overestimation factor* of 100%.

It is clear that the simple technique of overestimating the user-provided runtimes can reduce the sensitivity of our algorithms for runtime estimation errors to an acceptable level, with only a limited increase in cost. It should be possible for a decision support system to intelligently determine an estimated upper bound on the runtime estimation error, possibly even per user or per type of application. This upper bound can then be used in scheduling algorithms such as the one proposed in this contribution to achieve a higher degree of accuracy in their scheduling decisions. The development of appropriate algorithms to address this issue will be the focus of future work.

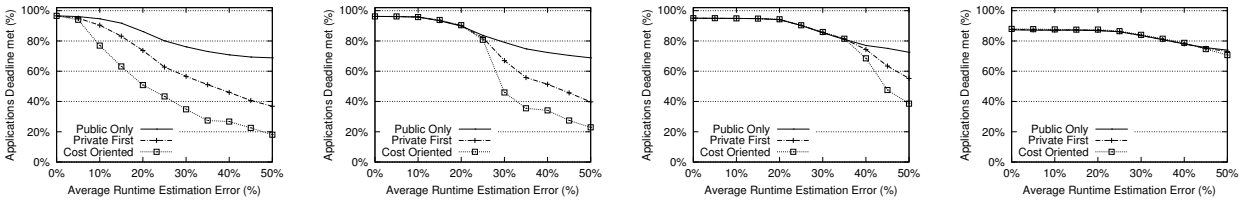
V. Related Work

In [10], a binary integer program is used to tackle the problem of selecting resources among different cloud providers. They focus on a static approach in which online applications –applications without deadline constraints, e.g. a web server– are scheduled on cloud providers in such a way that the total infrastructure capacity is maximized, given budget and load balancing constraints. Breitgand et al. [11] present a model for service placement in federated clouds, in which one cloud can subcontract workloads to partnering clouds to meet peak demands without costly over-provisioning. They use an Integer Program formulation of the placement program, and also provide a 2-approximation algorithm. Strebel et al. [12] propose a decision model for selecting instance types and workloads to evict from a local data center. They also use an optimization routine to create an optimal mix of internal and external resources for a given planning period. They assume a complete knowledge of the planning period, and include only one tariff in their optimization routine. In [13], a decision model is presented to determine bid prices on a resource market with varying prices, such as Amazon EC2's Spot Instances. The authors take into account the application's SLA constraints, and minimize the monetary cost for running the application. Kailasem et al. [14] propose a hybrid scheduling technique for a data-intensive document processing workload. For such a real-time workload, the transfer time to the external cloud is comparable to its computational time. They decide which job to outsource to the public cloud provider based on bandwidth and transfer time estimates, while fulfilling various queue-specific SLA constraints. They only consider settings with a single cloud provider and a fixed number of running instances, which is different in our contribution.

Other interesting initiatives in this domain can be found in the RESERVOIR [15] and OPTIMIS [16] projects.

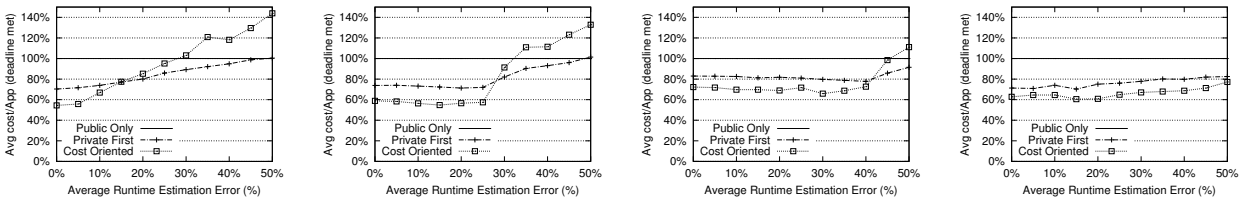
VI. Conclusion

We address the challenge of cost-efficiently scheduling deadline constrained batch type applications on hybrid clouds. In previous work [6], we presented a linear programming formulation of the optimization problem for a static set of applications. In this contribution, the problem is tackled by developing heuristics for cost-efficient scheduling that are able to operate on larger-scale problems. The proposed public and hybrid scheduling algorithms aim at providing an efficient solution with respect to the cost for running as much applications as possible within deadline constraints. The



(a) 0% (b) 10% (c) 50% (d) 100%

Figure 4. Impact of overestimation of *runtime estimation errors* on *Application deadlines met*.



(a) 0% (b) 10% (c) 50% (d) 100%

Figure 5. Impact of overestimation of *runtime estimation errors* on *Average cost per application*.

public scheduling component thereby takes into account data transfer costs and data transfer times. A hybrid scheduling mechanism is presented to take into account the *potential cost* of an application when scheduled on a public cloud provider. We demonstrate the results of all scheduling heuristics on workloads with varying characteristics. Results show that a cost-oriented approach pays off with regard to the number of deadlines met and that there is potential for a significant cost reduction, but that the approach is sensitive to errors in the user provided runtime estimates for tasks. We demonstrated the possibility to increase the conservativeness of the scheduler with respect to these estimates in order to deal with this issue without undermining the foundations of our cost-efficient scheduling algorithm. Future work in line with this contribution consists of partially redesigning the different components of our proposed solution to be aware of and more resistant to runtime estimation errors, as well as incorporating support for other public pricing models such as Amazon's *reserved* instances and *spot* markets.

REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2009.

[2] P. Mell and T. Grace, "The nist definition of cloud computing," National Institute of Standards and Technology, Information Technology Laboratory, Tech. Rep., 2009, version 15.

[3] D. Hilley, "Cloud computing: A taxonomy of platform and infrastructure-level offerings," Georgia Institute of Technology, Tech. Rep. GIT-CERCS-09-13, April 2009.

[4] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[5] M. D. Assunção, A. Costanzo, and R. Buyya, "A cost-benefit analysis of using cloud computing to extend thecapacity of clusters," *Cluster Computing*, vol. 13, pp. 335–347, 2010, 10.1007/s10586-010-0131-x.

[6] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *IEEE 3rd International Conference on Cloud Computing (CLOUD), 2010*, July 2010, pp. 228–235.

[7] P. Brucker, *Scheduling Algorithms*. Berlin Heidelberg: Springer-Verlag, 2004.

[8] D. England and J. B. Weissman, "Costs and benefits of load sharing in the computational grid," in *In Proceedings of JSSPP*, 2004, pp. 160–175.

[9] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "An evaluation of the benefits of fine-grained value-based scheduling on general purpose clusters," *Future Gener. Comput. Syst.*, vol. 27, no. 1, pp. 1–9, 2010.

[10] J. Tordsson, R. Montero, R. Vozmediano, and I. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Submitted for journal publication*, 2010.

[11] D. Breitgand, A. Maraschini, and J. Tordsson, "Policy-driven service placement optimization in federated clouds," IBM Research Division, Tech. Rep., 2011.

[12] J. Strebel and A. Stage, "An economic decision model for business software application deployment on hybrid cloud environments." in *Multikonferenz Wirtschaftsinformatik 2010*, M. Schumann, L. M. Kolbe, M. H. Breitner, and A. Frerichs, Eds. Göttingen: Universitätsverlag Göttingen, 2010, pp. 195 – 206.

[13] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, 2010, pp. 257 –266.

[14] S. Kailasam, N. Gnanasambandam, J. Dharanipragada, and N. Sharma, "Optimizing service level agreements for autonomic cloud bursting schedulers," in *International Conference on Parallel Processing Workshops*. Los Alamitos, CA, USA: IEEE Computer Society, 2010, pp. 285–294.

[15] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, and F. Galan, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, pp. 1 –17, 2009.

[16] A. J. Ferrer, F. Hernandez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. A. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, and et al., "Optimis: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66 – 77, 2012.