Cost-Efficient and Application SLA-Aware Client Side Request Scheduling in an Infrastructure-as-a-Service Cloud

Philipp Leitner, Waldemar Hummer, Benjamin Satzger, Christian Inzinger, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
{lastname}@infosys.tuwien.ac.at

Abstract

Providers of applications deployed in an Infrastructure-as-a-Service cloud permanently face the decision of whether it is more cost-efficient to scale up (i.e., rent more resources from the cloud) or to delay incoming requests, even though doing so may lead to dissatisfied customers and broken service level agreements. This decision is further complicated by the fact that not all customers have the same agreements, and not all requests require the same amount of resources devoted to them. In this paper, we present an approach for optimally scheduling incoming requests to virtual computing resources in the cloud, so that the sum of payments for resources and loss incurred by service level agreement violations is minimized. We discuss our approach based on an illustrative use case. Furthermore, we present a numerical evaluation based on real-life request data, which shows that our agreement-aware algorithm improves upon earlier work, which does not take service level agreements into account.

I. Introduction

Current research and practice of information systems engineering is seeing substantial interest in the idea of cloud computing [3], [5]. In cloud computing, resources, such as CPU processing time or disk space, are rented and released purely on demand. The advantages of this model are manifold, including potential cost savings for users, virtually limitless scalability, and greener IT because of reduced energy consumption. One popular approach to cloud computing is the Infrastructure-as-a-Service (IaaS) model [10], where virtual computing resources (virtual machines) are acquired and released on demand. This idea has been made popular in recent years by widely used implementations, such as Amazon's Elastic Compute Cloud[1] or the Eucalyptus[2] open source implementation. Even though the IaaS model delivers considerably more flexibility than previous server leasing models (e.g., renting a dedicated server at a commercial computing center), a certain level of planning remains necessary, as providers need to take billing time units [9] (BTUs) into account, the atomic measurement of billable computing resources. Typically, the BTU is relatively large (e.g., one hour for EC2) as compared to the time necessary to serve requests.

For applications deployed to IaaS clouds, service level agreements [7] (SLAs) are becoming increasingly important. SLAs are contractual agreements between application providers and their customers. They are negotiated on a per-customer basis (different customers typically have different SLAs), and govern the minimum quality (e.g., response time) that customers can expect. Violation of SLAs is problematic for application providers, as SLA violations incur direct or indirect financial losses for the provider. SLA-bound application providers face an important trade-off in IaaS settings. On the one hand, they want to minimize cases of SLA violations, as to minimize penalty payments and customer dissatisfaction, i.e., ultimately, to save money. On the other hand, the typical way to provide better QoS in an IaaS setting is to scale out [19], that is, to acquire a larger number of virtual computing resources from the cloud, which incurs costs as well. Alternatively, a provider may try to schedule application requests smartly among existing virtual computing resources, so as to fulfill the largest percentage of SLAs without paying for more resources. However, this decision is complicated by the fact that not all requests are equal (some requests take up more computing resources to complete), and neither are all SLAs (some customer's requests have tighter SLAs than

[1] http://aws.amazon.com/ec2/
[2] http://open.eucalyptus.com/

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.21

others, and some SLAs are more expensive to violate than others).

In this paper we propose an approach for scheduling application requests to IaaS virtual computing resources, so that SLA-bound service providers can minimize their overall costs. We define the total costs as the sum of the costs of SLA violations and IaaS virtual computing resources costs. The proposed approach takes all the previously mentioned intricacies into account. The main assumptions that are implied in this paper are that (1) application requests are stateless, i.e., requests can be scheduled to any computing resource without consideration of where earlier requests have been scheduled to, and that (2) the execution time of requests can be reasonably precisely approximated in advance by the application provider. Our approach does not assume that providers have any information about future requests.

The rest of this paper is structured as follows. Section II introduces an illustrative case, which exemplifies the setting of our approach. Section III contains the main contribution of the paper, the description of the SLA-aware request scheduling approach. We numerically evaluate our approach in Section IV, and discuss some related scientific work in Section V. Finally, Section VI concludes the paper, and discusses some important future research directions.

II. Illustrative Use Case

In the rest of the paper, we will use the case of BIaaS (pronounced *bias*), a Web-based business intelligence (BI) application for small and medium-sized enterprises as scenario. In essence, BIaaS allows smaller companies with no access to complex analytics to upload raw data (e.g., sales data) in an anonymized form to BIaaS via a secured Web service interface, in order to later on generate a number of standard or customer-specific BI reports (e.g., a predictive analysis of sales trends). BIaaS is run by a small startup company, which, by itself, does not own the significant processing power and storage necessary to generate these reports. Instead, BIaaS rents on demand virtual machines at a public IaaS cloud. Similarly, the necessary database cluster (to securely store the uploaded customer data) is maintained via a rented cluster of cloud data services.

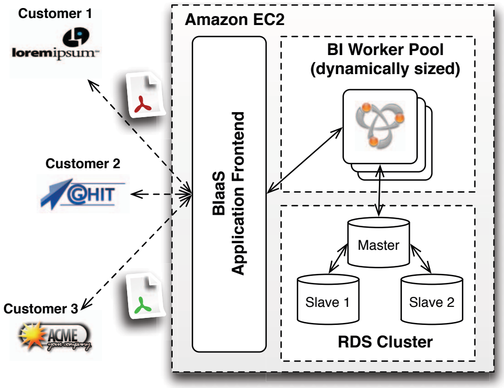

Figure 1. BlaaS Case Study Overview

This case is depicted in Figure 1. The BIaaS application consists of an application frontend, which hosts the Web services that customers use in order to interact with the application. This frontend is hosted on a single large EC2 cloud instance. The time-consuming process of report generation is delegated to one of a number of instances from a pool of workers. These workers are identical, and can be scaled up and down on demand, depending on current load. The actual customer data is stored in an Amazon Relational Database Service[3] (RDS) cluster, consisting of one master node and two slaves. In the figure, three example customers are depicted. Customer 1 has a very tight SLA with BIaaS, i.e., requests from Customer 1 need to be handled as soon as possible, as this SLA does not allow for much delay, and is very expensive to violate. Customer 3 also has an SLA with BIaaS, but this SLA is easier to maintain. Finally, Customer 2 does not have an SLA at all, i.e., requests from Customer 2 are served on a best effort basis.

The main challenge that we tackle in this paper is how to cost-optimally size the pool of workers (i.e., when to scale up and when to scale down), and how to optimally assign incoming requests to workers, depending on the current load of each worker and the SLAs and estimated durations associated with each request. This allows BIaaS to prevent over-provisioning, while at the same time maintaining compliance with the SLAs that are most important to the provider. Please note that we are illustrating the BIaaS case using Amazon cloud services mostly because of their high publicity. Our approach is in general not limited to any specific IaaS provider.

III. SLA-Aware Request Scheduling

In this section, we describe the main contribution of this paper, a cost-efficient and SLA-aware client side request scheduling technique. This approach can be used by cloud application developers to manage the pool of virtual resources in a way that is cost-efficient for the application provider.

A. Overview

In order to illustrate the decision problem tackled in this paper, Figure 2 shows a simple scheduling scenario based

[3] http://aws.amazon.com/rds/

on the BIaaS case.

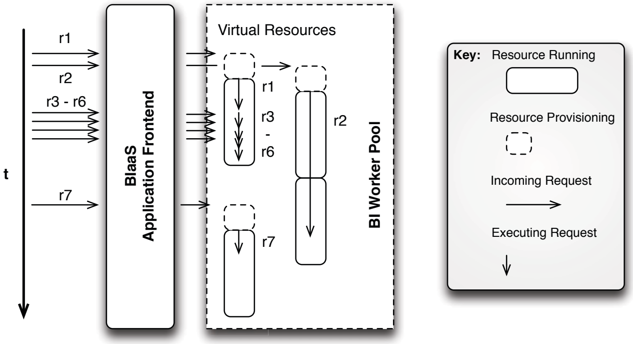

Figure 2. Scheduling Requests

In the figure, requests (r1 to r7) are arriving irregularly over time, and requests have individually different durations. Whenever a request is received by the BIaaS application frontend, it faces the decision of either instantiating a new virtual computing resource (and scheduling the request there), or scheduling the request to any of the existing virtual resources. r1, r2 and r7 trigger the allocation of new resources in the example. As indicated by the dashed start in the lifeline of each resource, during startup, there is a brief period of time when the resource is provisioned (i.e., the resource already exists, but is yet unable to process requests). The requests r3 to r6 are scheduled to existing resources. Note that requests can also be scheduled to resources which are currently busy, adding the request to any position in this resource's waiting list.

B. Assumptions

Our approach is based on some background assumptions. Most importantly, we assume that requests can be scheduled independently from each other. Essentially, this means that we require the individual requests to be stateless. State between requests may be maintained by the application, but this state needs to be stored in a way that is accessible by all virtual resources (e.g., in the RDS cluster in the BIaaS case). Additionally, in order to allow for meaningful scheduling, the application provider needs to be able to generate reasonable predictions of the duration of each request. Furthermore, we assume that requests, once started, should not be interrupted. Similarly, requests cannot be moved to a different virtual resource, once they have been scheduled. Note that these assumptions are in line with related earlier research work [9]. However, as compared to this earlier paper, we have relaxed the assumptions to the end that we do not consider virtual resources to be available instantly (i.e., we take the provisioning time of virtual resources into account).

C. Definitions

In order to formally describe our scheduling technique, we first need to define a number of relevant concepts. Firstly, a request $r \in R$ is issued by a customer. Requests have request durations $d_r \in \mathbb{R}$, which incorporate the times between when a request is received by the provider and the time the final result is delivered back. d_r consists of two important components, the waiting time $w_r \in \mathbb{R}$ (the time between receiving the request and starting to execute it) and the execution time $e_r \in \mathbb{R}$ (the time that it takes to actually carry out the request). The request duration is the sum of these components ($d_r = w_r + e_r$).

Requests have SLAs associated, which we refer to as $SLA_r \in SLA$. In our model, SLAs are functions that map request durations d_r to numerical costs, i.e., $SLA_r : \mathbb{R} \rightarrow \mathbb{R}$.

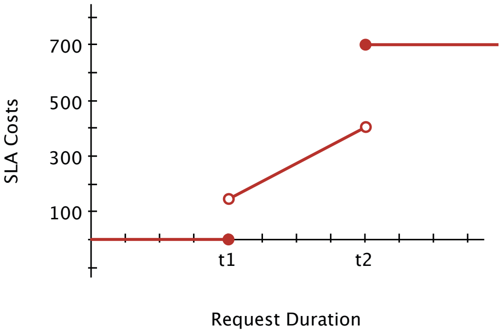

Figure 3. Example SLA Function

SLA functions come in many different shapes and forms [16]. In this paper, we generally assume a linear penalty function with two point discontinuities, similar to the example depicted in Figure 3. These SLA functions typically exhibit two important properties. Firstly, they are monotonically increasing, i.e., $\forall\, SLA_r \in SLA\, \forall d_1, d_2 \in \mathbb{R} : (d_1 < d_2) \implies (SLA_r(d_1) \leq SLA_r(d_2))$. This means that the penalty for a higher degree of SLA violation should never be smaller than the penalty for a "lesser" violation. Secondly, SLA functions have two special points, referred to as t_1 and t_2. Up to and including t_1 the penalty is 0 (no violation has occurred). Beyond this point, a penalty has to be paid which depends linearly on how much the request has been delayed, up to a second point discontinuity t_2. Starting with t_2, the customer and provider have essentially given up on the request. The request is cancelled, and a fixed penalty is charged.

However, SLAs are not the only cost factor that application providers need to keep in mind. Renting virtual resources also incurs significant costs. We refer to the virtual resources rented by an application provider as the set of H, consisting of concrete resources $h \in H$.

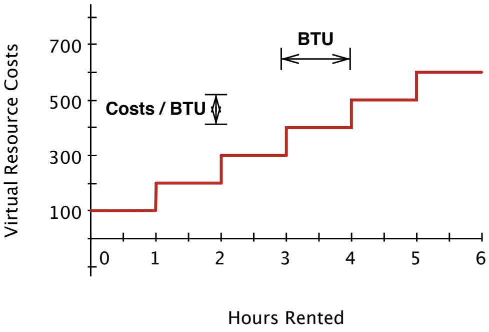

Figure 4. Virtual Resource Cost Function

Each resource has a request queue $Q_h \in Q$ (whenever a request is finished, the first request on Q_h is processed next). We use index terminology to refer to requests at specific positions in the queue, e.g., $Q_h[1]$ refers to the first (the currently executing) request in the request queue of h. The function $a : Q \rightarrow \mathbb{R}$ denotes the time that a currently executing request (by definition at the first index of the queue) is already executing. Furthermore, $|Q_h|$ denotes the queue length of the resource. The function $t : H \rightarrow \mathbb{R}$ captures the total time that a resource was running. Obviously, the costs of renting a given virtual resource h (referred to as $c_{vm}(t(h))$) are a function of that total run time $t(h)$ ($c_{vm} : \mathbb{R} \rightarrow \mathbb{R}$). In general, we assume c_{vm} to be a piecewise linear step function, with a structure similar to the one depicted in Figure 4. This type of function has two important parameters: (1) the length of the billing time unit btu, which is the granularity of billing for the resource (e.g., for Amazon EC2, virtual resources are billed by the hour, that is, btu is one hour); (2) the costs for a single virtual resource per BTU (c_{btu}). The function $f : H \rightarrow \mathbb{R}$ denotes the "free" time of a host, i.e., the time that we need to pay anyway, but for which currently no request is scheduled. Essentially, the free time is time that the host would be online but idle.

Finally, we need to keep in mind that virtual resources do not immediatly become available when they are rented. Much more, there is a startup phase, during which the resource is provisioned and booted. In our formal model, we use the constant p to refer to this provisioning time. p is defined as the time between requesting a new virtual resource from the cloud provider, and the time when the first request can be served by the resource. We assume that p is of comparable size for every virtual resource. Note that, in general, p is relatively large as compared to the execution time e_r for most $r \in R$. This means that launching a virtual resource on demand, to serve a given request, will often increase the total duration d_r of this request by orders of magnitude, as compared to using an idle existing virtual resource.

D. Request Scheduling

Based on the formal model described in Section III-C, we are now able to define an SLA-aware decision procedure for optimally scheduling incoming requests to virtual resources. In short, we aim to minimize the overall costs for the provider of the cloud application. We define the overall costs as in Equation 1. This equation can essentially be reduced to two vital components: reducing the sum of SLA violations for all requests, while at the same time keeping the operational costs for the virtual cloud resources minimal.

$$OC = \sum_{r \in R} SLA_r(d_r) + \sum_{h \in H} c_{vm}\left(t(h)\right) \rightarrow min! \qquad (1)$$

As indicated in Section III-A, for every request, scheduling can decide to either instantiate a new virtual computing resource or use an existing one. For each existing resource, we can insert the new request at any position in this resource's request queue. Consequently, for each request, we have to consider $1 + \sum_{h \in H}\left(|Q_h| + 1\right)$ different scheduling decisions. We can represent each scheduling decision as a 3-tuple $z =< r, h, i >$, with r being the request to schedule, h being the host to execute this request on, and i being the index of this host's queue that we want to place the request on. We propose the following simple greedy approach for deciding between these decisions: for a request r, select the scheduling decision where the costs $c_r(r, h, i)$ are minimal. For decisions to start a new virtual resource, the costs are as defined in Equation 2.

$$c_r(r, h, i) = SLA_r(p + e_r) + c_{vm}(p + e_r) \qquad (2)$$

Calculating the costs of scheduling to an existing host is more complex. Primitively, the costs in this case are the sum of three components (Equation 3), the SLA costs of this request, the potentially increased SLA costs of other requests affected by this decision, and the potentially increased cloud costs that this decision leads to.

$$c(r, h, i) = c_{SLA} + \Delta_{c_{SLA}} + \Delta_{c_{vm}} \qquad (3)$$

c_{SLA} can be defined straight-forward as the SLA penalty that has to be paid, under the assumption that the duration of the request is going to be the sum of the execution times of all requests scheduled before this one, plus the execution time of this request, minus the past execution time of the currently executing request (Equation 4).

$$c_{SLA} = SLA_r\left(\sum_{j=1}^{i-1}\left(e_{Q_h[j]}\right) + e_r - a(Q_h)\right) \qquad (4)$$

The increased cloud costs $\Delta_{c_{vm}}$ are defined as in Equation 5. If the host will be running idle for longer than what it takes to execute the request, the request can essentially be handled for free. Otherwise, we need to factor in the costs of renting this resource for one or even more additional BTUs.

$$\Delta_{c_{vm}} = \begin{cases} \left| \frac{e_r - f(h)}{btu} \cdot c_{btu} \right| & \text{if } e_r > f(h) \\ 0 & \text{if } e_r \leq f(h) \end{cases} \tag{5}$$

Finally, the sum of all changes to the SLA costs of other requests scheduled to the same host is defined in Equation 6. For simplicity, we use d_r to refer to the originally predicted duration of an existing request here. Essentially, this equation signifies that, for each request which is going to move backwards one slot in the queue, we need to calculate and sum up the new SLA costs minus the original SLA costs prior to the change.

$$\Delta_{c_{SLA}} = \sum_{j=i}^{|Q_h|} \left(\begin{array}{l} SLA_{Q_h[j]}\left(d_{Q_h} + e_r\right) - \\ SLA_{Q_h[j]}\left(d_{Q_h}\right) \end{array} \right) \tag{6}$$

If two or more decisions are associated with the same costs, we need an additional tie breaking criterion. To this end, we may either select one of these decisions at random, or use a principle that we refer to as "minimal reduction of possibilities". This principle prefers decisions, which use the shortest idle times that are still suitable to handle the request. That is, costs being equal, we prefer to use the smallest block of resources available, so that larger blocks of idle time remain available to handle possible future requests.

E. Releasing Virtual Computing Resources

Besides these scheduling decisions, the application frontend in the overview in Figure 2 also needs to decide when to release unused existing resources. However, without information about future requests, resource release decisions are trivial. In this case, the cost-optimal strategy for the provider is to never release resources until the end of the next BTU (i.e., if the BTU is one hour, never release resources until another full hour has passed, as this remaining time needs to be paid anyway), and to always release resources which are not in use at the end of a BTU.

With more information about the request distribution, it is possible to construct release heuristics, which improve upon this trivial strategy (for instance, leave one or more unused "spare" resource available to prepare for upcoming load peaks), but the performance of such heuristics depends strongly on the costs of keeping virtual computing resources online versus the provisioning time p of new resource, i.e., in tendency, keeping spare resources becomes more attractive if resources are cheap and p is large. We leave further discussion of resource release heuristics to future research.

IV. Evaluation

In this section, we numerically evaluate the SLA-aware scheduling approach. Therefore, we have implemented the approach, as well as a number of other scheduling algorithms, using a test harness based on the Groovy[4] scripting language. We compare scheduling algorithms based on a real-life request log, courtesy of the Grid Workloads Archive [11], which we have associated with generated SLA data.

A. Comparison Algorithms

In order to illustrate the usefulness of our approach, we will compare our SLA-aware greedy scheduling strategy with some established algorithms, as surveyed in [9]. More concretely, we have implemented the comparison algorithms summarized in Table I.

Comparison Algorithm	Reasoning
1VM4All	Provides a lower bound on cloud costs
1VMPerReq	Maximum parallelization of requests
BinPacking	Maximizes utilization of resources

Table I. Summary of Comparison Algorithms

The 1VM4All scheduling algorithm uses only a single virtual resource to handle all requests. All incoming requests are appended to the end of the queue of this single resource. 1VMPerReq is the other extreme. This algorithm does not enqueue any requests. Instead, if an idle running resource is found, this idle resource is used. If no idle virtual resource is available, a new one is requested and used. Finally, we also use a classic heuristic for online bin packing [6], best fit bin packing. Essentially, this heuristic schedules a request to the running machine whose free time best matches the execution time of the request. This algorithm aims at maximizing the utilization of virtual resources.

B. Experimental Setup

For a fair comparison of the algorithms, we have used a sample of a real-life data set collected by the Grid Workloads Archive. More concretely, we have used the AuverGrid workload trace with the identifier *Gwa-t-4*[5].

[4] http://groovy.codehaus.org/

[5] http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Workloads.Gwa-t-4

From this trace, we have skipped the first 1000 requests (as these seem to present a relatively uninteresting bootstrap phase of low activity), and used the next 2000 requests with a processing time between 10 and 60000 seconds. From these requests, we extracted their duration as well as the delay between requests, and associated randomly generated SLA functions of the form depicted in Figure 3. In this way, we have extracted two different data sets, one with rather lenient SLAs (i.e., the SLAs are defined with significant leeway for delays) and one with stricter SLA definitions. For the lenient data set, the average violation threshold (t_1 in Figure 3) is set to $2.5 \cdot e_r$, while t_1 for the strict set is in average only $1.5 \cdot e_r$. We provide both experimental base data sets online[6], to allow for scientific validation of our results.

C. Results

In the following, we will compare the SLA and cloud costs accruing for the application provider when scheduling requests based on the different algorithms. However, before looking at final costs, we present some important insights into the runtime of our experiments, which exemplify better what is going on during each experiment run. The following plots depict runs against the data set with more lenient SLAs, however, the general insights described below are equally valid for the other data set. Furthermore, we have omitted 1VM4All in these plots, as the special nature of this algorithm leaves plotting the number of hosts and requests uninteresting.

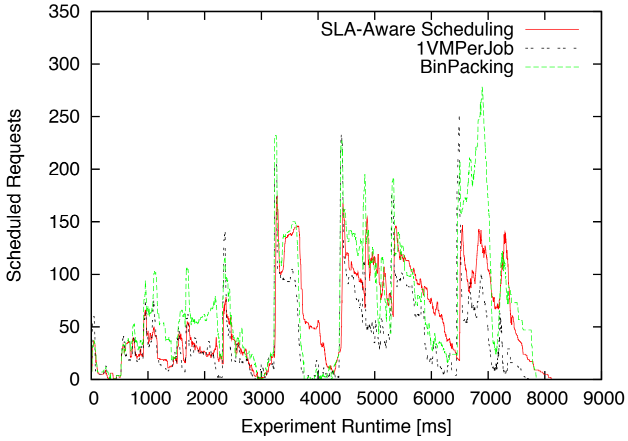

Figure 5. Scheduled Requests

Firstly, Figure 5 depicts the number of currently scheduled (i.e., executing or enqueued) requests at each point in time during the experiment. Waiting times are minimal for 1VMPerJob (requests are per definition only waiting while new virtual resources are provisioned). As BinPacking aims at making the most out of already paid resources, some requests are delayed longer in order to maximize utilization. Our SLA-aware scheduling approach is positioned in the middle, as it aims at enqueuing requests with more lax SLAs, and tries to handle expensive SLAs directly.

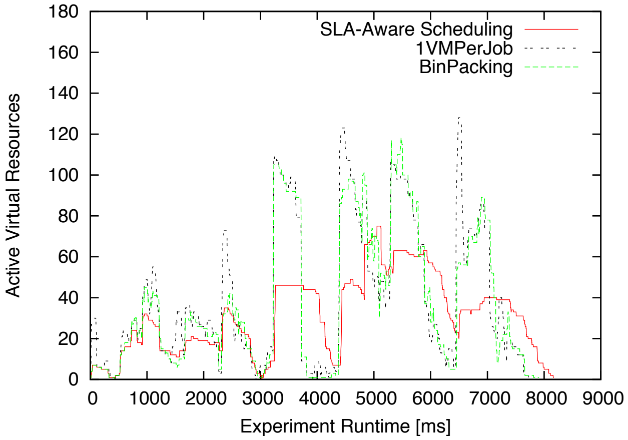

Figure 6. Active Virtual Resources

Even more interesting is Figure 6, which provides a view at the number of currently active virtual resources at each point in time. Evidently, our SLA-aware approach is more "stable" in its resource usage, while the number of active resources fluctuates more drastically using both 1VMPerJob and BinPacking with current load. As the provisioning time p of resources is typically significant as compared to the execution time e_r of requests, we consider this relative stability to be a useful property of the approach.

Algorithm	SLA Costs	Cloud Costs	Overall Costs (OC)
SLA-Aware	883	255400	256283
1VM4All	1756684	230600	1987284
1VMPerJob	0	321000	321000
BinPacking	3171	305100	308271

Table II. Experimental Results (Lenient SLAs)

Obviously, the most interesting question is whether our SLA-aware approach is able to reduce the overall costs OC as compared to other algorithms. Hence, final costs (summed up over all requests in the data sets) for all algorithms and both data sets are listed in Table II and Table III. As we can see, our SLA-aware approach leads to the least overall cost for both data sets. As expected, 1VMPerJob minimizes SLA costs, but does so by overpaying on cloud resources. 1VM4All minimizes cloud costs, but SLA payments are larger by orders of magnitude than with all other competitors (in fact, many requests even exceed t_2, and, hence, are in fact cancelled).

[6] http://www.infosys.tuwien.ac.at/prototypes/VRESCo/cloud12/data

BinPacking provides a middle ground between SLA and cloud costs, but overall costs are still higher than using our SLA-aware approach.

Algorithm	SLA Costs	Cloud Costs	Overall Costs (OC)
SLA-Aware	32023	253400	285423
1VM4All	12480685	230500	12711185
1VMPerJob	8338	320600	328938
BinPacking	46285	304700	350985

Table III. Experimental Results (Strict SLAs)

V. Related Work

In recent years, cloud computing [3], [5] has become a proliferating research area, with many important results being generated around the globe. Generally speaking, many current cloud computing research approaches are cost-aware [17], [24], in the sense that they take the costs of renting virtual resources into account. Typically, the base assumption is that an externally fixed number of requests needs to be handled within a fixed time frame, with the only variable being the cloud deployment and scheduling. This is unlike our work, which allows to find the best tradeoff between request execution performance and costs for the cloud.

Our main vehicle for bringing application performance and cloud hosts into comparable dimensions are service level agreements (SLAs). SLAs are an active research topic within the services computing area, leading to specifications, such as WSLA [7] or WS-Agreement [2]. Many approaches exist for managing SLAs in service-based environments (e.g., [23]), including approaches that also take costs of adaptation and SLA violations into account [15]. The latter paper is particularly interesting for us, as the formalization that we have used to model the execution duration / infrastructure cost tradeoff in Section III is based on this earlier work. In the area of cloud computing research, [1] provided some ground work on SLAs in clouds, including a conceptual framework and some typical service level objectives. Unfortunately, [1] does not take application and customer-specific SLAs into account, which makes it less relevant for the purposes of the current paper. The short paper presented in [20] is more relevant, starting from a similar idea as the contribution of the current paper. However, instead of finding the optimal tradeoff between SLA violations and cloud costs, [20] aims for the best middle ground between SLA violations and revenue generated by the cloud application.

Conceptually, the current paper is based on the ideas first presented in [9]. This paper introduced various simple algorithms for scheduling requests to virtual computing resources. We extend upon this idea by including the notion of SLAs, and present a novel, SLA-aware decision procedure. The evaluation, that we have presented in Section IV, mostly compares the results of our decision procedure to these earlier results. However, [9] was not the first paper to discuss request scheduling in the cloud. Other contributions in this area include [18], which works based on queueing theory, [21], which applies the notion of gang scheduling (a scheduling approach stemming from Grid job scheduling), and [13], which introduces a new algorithm called DPSA (dynamic priority scheduling algorithm). These papers mainly focus on the scheduling problem that the cloud operator faces (for instance, scheduling virtual computing resources to physical hosts). Approaches that aim at scheduling at client-side, as we do, are currently harder to come by. One interesting approach is discussed in [4], where client-side scheduling onto a combination of in-house and cloud resources has been researched. Additionally, [9] as described above, also covers client-side cloud scheduling. Unlike our contribution, these papers do not explicitly cover SLAs of application providers.

Finally, some interesting work orthogonal to the contributions of this paper needs to be discussed. [12] presents a GNU Octave based implementation of performance predictions in a grid or cloud. In the context of our work, this is particularly relevant, as it allows application providers to generate the necessary predictions of request execution times, which we have excluded in Section III. Note that other approaches, such as artificial neural networks [14] or finish time prediction [8] may also be suitable to generate projections of execution times. [22] is interesting, as it provides a cloud-based middleware, which automatically adapts to incoming load to improve performance. However, this paper does not consider SLAs or the costs of scaling up and down. Similarly, [16] presents CloudScale, another cloud computing middleware that transparently handles scaling for the application provider. The approach discussed in the current paper has been designed to be easily usable in the scope of CloudScale.

VI. Conclusions

We have presented an application SLA-aware approach for scheduling requests to virtual computing resources in an IaaS cloud. Essentially, our work resolves the request execution time / infrastructure costs trade-off described in [9] by breaking down both factors to costs for the provider, hence allowing us to find an optimal solution to this scheduling problem. We presented an easy-to-implement cost-based decision procedure, and evaluated our approach based on a real-life request data set from the scientific computing domain. We have numerically compared our decision procedure with a set of algorithms published earlier, and found that our approach leads to better results (i.e., lower costs for the application provider) than any of these.

A. Future Research Directions

In this paper, the main focus was on the solution of the SLA-aware scheduling problem itself, abstracting away from some important technical issues. In earlier work, we have presented the CloudScale framework [16] for transparently scaling applications in the cloud. As next steps, we plan to integrate the scheduling approach discussed here into CloudScale, and, consequently, to evaluate our ideas using a more extensive case study, which features a real running cloud application. Furthermore, we plan to relax some of the assumptions of the current iteration of our work. More concretely, we will investigate possibilities to pause and migrate executing requests within CloudScale, allowing us to "update" assignments of requests to virtual computing resources later on, even after a request has started to execute. We plan to investigate the usage of machine learning based prediction technqiues to implement concrete tools for projecting the execution time e_r of requests in advance, as required by our scheduling approach. In earlier work, we have already applied artificial neural networks for conceptually similar problems [14]. Finally, as already indicated in Section III-E, a more detailed investigation of resource release heuristics, which incorporate predictions about future request distributions, will also be part of our future work.

Acknowledgement

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreements 215483 (S-Cube) and 257483 (Indenica).

References

[1] M. Alhamad, T. Dillon, and E. Chang, "Conceptual SLA Framework for Cloud Computing," in *4th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, 2010.

[2] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web Services Agreement Specification (WS-Agreement)," Open Grid Forum (OGF), Tech. Rep., 2006, http://www.gridforum.org/documents/GFD.107.pdf, Last Visited: 2012-01-29.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[4] M. D. Assunção, A. Costanzo, and R. Buyya, "A Cost-Benefit Analysis of Using Cloud Computing to Extend the Capacity of Clusters," *Cluster Computing*, vol. 13, pp. 335–347, 2010.

[5] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computing Systems*, vol. 25, pp. 599–616, 2009.

[6] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation Algorithms for Bin Packing: a Survey*. PWS Publishing Co., 1997, pp. 46–93.

[7] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, "Web Services on Demand: WSLA-Driven Automated Management," *IBM Systems Journal*, vol. 43, no. 1, pp. 136–158, 2004.

[8] B. F. Dongen, R. A. Crooy, and W. M. Aalst, "Cycle Time Prediction: When Will This Case Finally Be Finished?" in *OTM Confederated International Conferences*, 2008, pp. 319–336.

[9] S. Genaud and J. Gossa, "Cost-wait Trade-offs in Client-side Resource Provisioning with Elastic Clouds," in *4th IEEE International Conference on Cloud Computing (CLOUD 2011)*, 2011.

[10] D. Hilley, "Cloud Computing: A Taxonomy of Platform and Infrastructure-Level Offerings," 2009.

[11] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema, "The Grid Workloads Archive," *Future Generation Computer Systems*, vol. 24, pp. 672–686, 2008.

[12] G. Kousiouris, D. Kyriazis, K. Konstanteli, S. Gogouvitis, G. Katsaros, and T. Varvarigou, "A Service-Oriented Framework for GNU Octave-Based Performance Prediction," in *2010 IEEE International Conference on Services Computing (SCC'10)*, 2010, pp. 114–121.

[13] Z. Lee, Y. Wang, and W. Zhou, "A Dynamic Priority Scheduling Algorithm on Service Request Scheduling in Cloud Computing," in *International Conference on Electronic and Mechanical Engineering and Information Technology (EMEIT)*, 2011, pp. 4665–4669.

[14] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime Prediction of Service Level Agreement Violations for Composite Services," in *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'09)*, 2009.

[15] P. Leitner, W. Hummer, and S. Dustdar, "Cost-Based Optimization of Service Compositions," *IEEE Transactions on Services Computing (TSC)*, 2012, to appear.

[16] P. Leitner, W. Hummer, B. Satzger, C. Inzinger, and S. Dustdar, "CloudScale - a Novel Middleware for Building Transparently Scaling Cloud Applications," in *ACM Symposium on Applied Computing (SAC)*, 2012.

[17] J. Li, S. Su, X. Cheng, Q. Huang, and Z. Zhang, "Cost-Conscious Scheduling for Large Graph Processing in the Cloud," in *IEEE International Conference on High Performance Computing and Communications (HPCC'11)*, 2011, pp. 808–813.

[18] L. Li, "An Optimistic Differentiated Service Job Scheduling System for Cloud Computing Service Users and Providers," in *3rd International Conference on Multimedia and Ubiquitous Engineering*, 2009, pp. 295–299.

[19] M. M. Michael, J. E. Moreira, D. Shiloach, and R. W. Wisniewski, "Scale-Up x Scale-Out: A Case Study Using Nutch/Lucene," in *21th International Parallel and Distributed Processing Symposium (IPDPS 2007)*, 2007, pp. 1–8.

[20] H. J. Moon, Y. Chi, and H. Hacigümüs, "SLA-Aware Profit Optimization in Cloud Services via Resource Scheduling," in *Proceedings of the 2010 6th World Congress on Services (SERVICES'10)*, 2010, pp. 152–153.

[21] I. A. Moschakis and H. D. Karatza, "Performance and Cost Evaluation of Gang Scheduling in a Cloud Computing System With Job Migrations and Starvation Handling," in *16th IEEE Symposium on Computers and Communications (ISCC 2011)*, 2011, pp. 418–423.

[22] B. Satzger, W. Hummer, P. Leitner, and S. Dustdar, "ESC: Towards an Elastic Stream Computing Platform for the Cloud," in *4th IEEE International Conference on Cloud Computing (CLOUD 2011)*, 2011, pp. 348 – 355.

[23] A. Schmietendorf, R. Dumke, and D. Reitz, "SLA Management - Challenges in the Context of Web-Service-Based Infrastructures," in *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, 2004.

[24] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh, "A Cost-Aware Elasticity Provisioning System for the Cloud," in *31st International Conference on Distributed Computing Systems (ICDCS'11)*, 2011, pp. 559–570.