COST-AWARE ON-DEMAND RESOURCE PROVISIONING IN CLOUDS

Completed Research Paper

Bendler, Johannes
University of Freiburg, Chair for Information Systems Research
79098 Freiburg, Germany
johannes.bendler@is.uni-freiburg.de

Neumann, Dirk
University of Freiburg, Chair for Information Systems Research
79098 Freiburg, Germany
dirk.neumann@is.uni-freiburg.de

Abstract

Cloud computing has been gaining increased popularity among companies for a couple of years. Low upfront costs and the enormous elasticity make it an attractive alternative to in-house IT solutions. However, most companies still tend to overprovision their IT infrastructures induced by the buffering behavior to avoid violations of service level agreements. The introduction of dynamic scaling in IT infrastructures promises enormous cost-saving potential. We identify factors of influence for cost-aware operation of an on-demand provisioned system and propose a novel provisioning model that attempts to optimally scale the cloud at any time. Following the design science paradigm we formulize the elements of our simulation framework and carry out extensive simulations under fine-grained real world settings. Compared to common operation strategies, our approach delivers superior performance results. Finally, we derive managerial implications based on the cloud customer’s preference between cost-awareness and SLA compliance.

Keywords: Cloud Computing, Simulation and Modeling IS, Information Systems Management
Introduction

Computing infrastructures are evolving at a remarkable rate due to their high availability over the Internet and their swift scalability capacities during runtime. Accessibility and scalability are only few characteristics that make information systems a critical success factor for many businesses. The high dynamics associated with the ongoing evolution of IT opened the door for new corporate and commercial opportunities. The decrease of processing and response times of IT infrastructures allows the development of powerful services and interfaces that handle massive amounts of data – a trend which is commonly referred to as ‘Big Data’.

While in the past IT infrastructures tended to use monolithic architectures, the paradigm has shifted towards distributed systems. Clouds represent state-of-the-art in distributed systems – they aggregate heterogeneous hardware from different locations to provide a solid platform for applications built on top. As such, information systems are frequently becoming hardware independent and provide high flexibility. Providers can easily extend the cloud by adding additional hardware during runtime, accommodating the customers demand for seemingly endless computing resources. However, this trend towards flexible demand and endless resources implies an increasing importance on the scaling of cloud systems to cope with sudden spikes in demand for computing resources. The costs for a rented cloud system strongly relate to the cloud customers’ ability to scale the system up or down instantaneously to exactly fit the very current demand. Thus, there is a need for cloud management capabilities that can deliver such flexible scaling autonomously. If scaling involves long latency times, either too many resources are maintained, or the available resources are too few to accommodate the increased demand. The right scaling of clouds is still an open research issue, since there are many complexities arising from the infrastructures and from the users that need to be served. The increase in demand of computing resources requires higher ecological awareness of the environmental impact.

Today, utilizing cloud services refers to the on-demand provisioning of computing resources. Enterprises, no matter their size, can relay their entire computational demand to the cloud and have it automatically scale according to their needs. The outsourcing of computational requirements to cloud computing renders long-term investments in hardware and administrative efforts obsolete. However, many cloud management systems still greatly overprovide resources to avoid the anger of their customers for which SLAs have been violated. As a consequence, enterprises tend to statically provision their cloud resources to cover load spikes, even though cloud computing offers much greater flexibility. Exploiting the elasticity of clouds, enterprises can dynamically provision resources to fit the actual workload rather than eventual peaks. Volatile workload and varying task characteristics exhibit complex challenges in operation but offers high potential of cost-savings at the same time. The opportunity of cost-savings and increased resource efficiency are the incentives of our novel cloud management framework proposed in this work.

In this paper, we argue that using Big Data, even complex infrastructures can be fairly well understood by the vast amounts of observations on the system performance. If properly processed, the data can potentially help cloud operators to more efficiently scale the cloud resources. Within the scope of this research, we introduce the novel cloud management framework that can make use of the observational data to push the resource consumption of clouds closer towards the (theoretical) optimum, which coincides with only little overprovisioning. Our cloud management framework improves on related research by Hedwig et al. (2012) by emphasizing the use of heterogeneous resources.

We define novel cost-aware provisioning policies capable of decreasing a cloud customer’s operational costs. To our best knowledge, cost-aware provisioning policies for cloud computing have not been formulated and evaluated before. Each of our cost-aware policies is tested against Amazon EC2 instance prices using simulation with real-world distributions of task characteristics. We are following the design science paradigm by formally describing the elements of the framework so that there are no ambiguities of our approach that hamper an implementation in the real world (Hevner et al. 2006).

Thus, the main contributions of this research work unfold as follows. Our research provides:

- Novel cost-aware provisioning policies for cost reduction, formally described;
- Maximizing SLO compliance while minimizing costs;
- Paying respect to customers’ tradeoff preference between SLOs and costs;
- Extensive simulation to analyze and evaluate the proposed policies;
The remainder of this paper is structured as follows. In the first section, we identify and analyze factors that influence cost-aware cloud operation. We carve out the design problem domain and formulate three main requirements of the cost-aware provisioning policy. Subsequently, we discuss the state-of-the-art in the research areas that are combined in our interdisciplinary model. We then propose and formally define the cost-aware provisioning model and its four core components. In the evaluation section, we present the simulation environment and the metrics for performance rating of provisioning policies. Subsequently, we present the simulation results and their managerial implications. Finally, we deliver a conclusion of our work.

**Design Problem Statement**

According to the report of Toronto City Summit Alliance (2010), most servers in datacenters only operate at approximately four per cent utilization. This finding presents clear evidence that most enterprises tend towards overprovision of their information systems. The reason for this overprovisioning stems from the fact that the associated workload is generally volatile. Enterprises avoid taking the risk of running into a supply shortfall of computing capacities. However, the workload exhibits periodic patterns daily, weekly or monthly. The constant overprovision of IT systems satisfies all computational demands at any time, but at very high costs. A method to dynamically scale the IT system during runtime that is based on current or expected workload with respect to resource costs can greatly reduce these costs. Moreover, a cloud provider may even be able to serve more customers at the same time, because his customers keep less virtual instances (resources) reserved without actually using them. Once the demand for computing resources is exactly known, the cloud customer may select the optimal number and size of virtual instances that best satisfies the user requirements. For example, if the customer runs a service on the cloud but is short on computing power, the system may be scaled up by selecting a more powerful virtual instance.
- **System Scale:** The system scale has the highest impact on a cost-aware operation, because it is directly related to generated costs. The theoretical cost-saving potential is sketched in Figure 1. The solid black graph represents an exemplary workload over a certain period of time; the dashed line at the top of the figure reflects the system state, which is statically provisioned to accommodate all eventual peak loads. The information system then is over-provisioned in all off-peak periods, which prevail for most of the time. The sketched area between the actual workload graph and the over-provisioned system configuration could theoretically be cut down by means of a dynamic on-demand provisioning.

- **Service Level Agreements:** A service level agreement (SLA) defines a contract between provider and consumer of an information system. Bilateral SLAs can be characterized by the definition of distinct objectives (SLOs), such as response time or error ratio. If all objectives are fulfilled, the overall SLA is considered satisfied. Besides the SLOs, an SLA conditions the treatment of over and under-satisfaction of stipulated services. Figure 3 depicts the typical concept of an SLA and its interplay with SLOs. The superior SLA contains and frames various SLOs in a contractual way, bound to valuations and penalties. Thus, satisfaction of an SLA usually generates positive cash-flow while its violation results in a contracted penalty. The responsibility to fulfill the SLO is mostly transferred to the provider, if computation is outsourced to the cloud. Since the cloud customer decides on the amount and scale of rented resources, influence on SLO satisfaction partially resides at his hand. Especially the swiftness of computations relayed to the cloud is affected by the scale of the rented cloud resources. For this reason, finishing computing tasks within a certain amount of time is considered to be the most low-level SLO for scaling decisions.

![Figure 3. Components of Service Level Agreements](image)

- **Characteristics of Tasks and Workload:** Computational demands are manifold ranging from short tasks like a simple website request to large and complex requests for data analysis or scientific computing. On a cloud platform, every imaginable workload may be possible. The computational power offered by a cloud provider is usually adaptable to any kind of task. Along with the task characteristics, the workload process is an important factor on the ability for reaching a target SLO. The low mean utilization of servers in datacenters hypothesizes that workload processes are highly volatile. Gmach et al. (2007) extract typical weekly patterns of workloads for datacenters. The rationale of these patterns is taken into account for the generation of workloads in this work. The cost-aware provisioning model requires stable and reliable workload forecasts. Workload forecasting is a dynamic research field and various methods have been developed (Urgaonkar et al. 2008; Gmach et al. 2007; Powers et al. 2005; Khan et al. 2012). This work applies an approach consisting of a long-term trend based on historical data, a seasonal part with known period length and a residual part, which can be estimated using a Yule-Walker Forecast.

- **Characteristics of Virtual Instances:** Most cloud providers offer on-demand resource provisioning for both public and private clouds. Cloud customers are able to enjoy a high degree of flexibility regarding their demand for computational power. Though there are options to rent exclusive hardware or to scale the system using spot instances (Amazon Web Services 2013). Most commonly, providers offer virtual instances that are paid per hour of computation. Thus, a cloud customer who is facing high workload can simply scale the system up by dynamically requesting further instances from the provider. Instances are offered in various configurations and prices. For
example, Amazon offers sixteen different instance configurations in its on-demand sector of EC2. Instances differ in the capacities of supplied resources (memory, computing units, storage size, IO-capabilities) and pricing additionally takes the requested operating system into account. The prices are not scaled linearly along the capacity of provided resources, as shown by the exemplary choice of several different Amazon EC2 on-demand instances in Table 1. The selection by minimum price per computing unit differs from the optimal selection by minimum price per memory capacity. However, each of these choices provide at least some capacity of the respective other resources. The effect of available resource capacities based on a selected instance type increases the problem complexity significantly. The variations in price per computing unit and price per memory between different instances deliver the opportunity to find a cost-optimal alignment of rented instances.

<table>
<thead>
<tr>
<th>Instance name</th>
<th>Price per hour in USD</th>
<th>Computing units</th>
<th>Memory in Gigabytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Amount</td>
<td>Price/Amount</td>
</tr>
<tr>
<td>M1 Small</td>
<td>0.065</td>
<td>1</td>
<td>0.065</td>
</tr>
<tr>
<td>M1 Medium</td>
<td>0.130</td>
<td>2</td>
<td>0.065</td>
</tr>
<tr>
<td>M1 Large</td>
<td>0.260</td>
<td>4</td>
<td>0.065</td>
</tr>
<tr>
<td>M3 XLarge</td>
<td>0.550</td>
<td>13</td>
<td>0.042</td>
</tr>
<tr>
<td>M3 2XLarge</td>
<td>1.100</td>
<td>26</td>
<td>0.042</td>
</tr>
<tr>
<td>M2 XLarge</td>
<td>0.460</td>
<td>6.5</td>
<td>0.071</td>
</tr>
<tr>
<td>C1 Medium</td>
<td>0.165</td>
<td>5</td>
<td>0.033</td>
</tr>
</tbody>
</table>

In this work, we propose a cost-aware provisioning model that takes all factors of influence into account to provide an informed decision on the system scale. The positioning of our model is depicted in Figure 4. Additionally, we will later introduce the cloud customer's tradeoff preference as another factor of influence.

Related Work

The cost-aware provisioning model is influenced by various research. This section presents the current state-of-the-art in the different research of performance modeling, workload forecast, SLA management and adaptive information systems. A well-known publication in the field of performance analysis and modeling is for example Urgaonkar et al. (2008), where queuing models are applied to automatically allocate resources in information systems. A model of performance characteristics using machine learning is provided by Cohen et al. (2004). Both models can be utilized when it comes to multi-tier applications. The reactive migration controller developed in Gmach et al. (2009) applies for single-tier systems, similar to work in this research paper, but focuses on migration for savings in power consumption. Their approach aims at optimal load distribution and reduction of required hardware, while our paper presents a model that focuses on cost-awareness during the on-demand provisioning process. Further work involving queuing models is provided by Chandra et al. (2003), who developed a resource allocation model for shared datacenters as well as a time series based mechanism to forecast workload. During provisioning, they do not consider compliance of service level objectives. The automated control model for virtual resources by Schröder-Preikschat et al. (2009) dynamically manages the resources assigned to virtual machines or
completely migrates entire virtual instances. The ability of live migration is one of the largest improvements in state-of-the-art cloud systems, but it does not assist on the consumer side. Ardagna et al. (2007) propose a model for resource management of multiple concurrent systems. They optimize the model for certain points in time and do not observe the long-term SLO compliance. Schulz (2010) highlights an approach of measuring partial fulfillment of service level agreements. The surge protection model for dynamic resource allocation developed in Lassette et al. (2003), as well as the autonomic control model for elastic storage scaling based on utilization by Parashar et al. (2010) deliver interesting insights and applicable methods, but both focus on the provider side. An aspect that is applicable on the consumer side is admission control as proposed by Bichler and Setzer (2007). If the faced workload exceeds the capacity of a system scaled to a maximum, informed decisions on task admission have to be made. Our recent research has transferred the admission control approach to cloud computing in Bendler et al. (2013).

The impact of volatile workload on system behavior and stability is a very important aspect for computing environments that aim to be adaptive to the computational demand. A general overview of workload and its modeling is provided by Feitelson (2005). Also research by Urgaonkar et al. (2008) contains a workload forecast model based on empirical estimation of historic workload processes. An alternative approach is the application of Fourier transformations to smooth workload as provided by Gmach et al. (2007). In their research the workload prediction is performed by time series analysis. Another approach that uses time series analysis and additionally regression for workload prediction is developed in the paper by Powers et al. (2005). Khan et al. (2012) provide a workload prediction model based on multiple time series. All four methods are possible alternatives to refine the basic forecast used in our work.

In relation to service level agreements, there exists very high research activity. Buyya et al. (2009) deliver a broad overview of service level agreements in the scope of cloud computing. Alhamad et al. (2010) identify a classification of SLA characteristics for the application in clouds. An overview of aggregation of low-level metrics to high-level SLAs is proposed by Emeakaroha et al. (2010). Deeper research dealing with risk analysis based on service level agreements by Yeo and Buyya (2007) additionally determines the economic feasibility of jobs. The economic aspect is also regarded in the policy driven refinement by Aib and Boutaba (2007). Both of them focus on batch processing but do not cope with dynamic workload and uncertainty. Approaches of automatic negotiation of SLAs are stated by Hasselmeyer et al. (2006) and Stantchev and Schröpfer (2009). The latter one also takes QoS into account. An architecture for autonomous management of QoS based on SLA specification is proposed by Koller and Schubert (2007). Similarly, Buco et al. (2004) developed an SLA management system based on business objectives. Hedwig et al. (2012) provide an approach to risk-aware design of SLAs. Finally, Sahai et al. (2002) designed a general scheme for service level agreements that enables autonomic management.

The Cost-Aware Provisioning Model

Based on the analysis of critical influence factors, the cost-aware provisioning model was designed. The general information flow and the four internal main concepts are outlined in Figure 5. The model aims to find a nearly cost-optimal operation strategy for renting on-demand cloud resources while maintaining a minimum level of SLO compliance in long-term. This operation requires a tradeoff between system performance and resource cost.

![Figure 5. Cost-Aware Provisioning Model with Internal Components](image-url)

The cost-aware provisioning model makes informed decisions towards system scaling based on the current system scale and system load, a target service level agreement, assumptions of task requirements and
distributions in a workload model, as well as available virtual instance configurations and their prices. Due to the complex environment, we test the model’s performance in a simulation framework. The model as well as the simulation setup is designed highly modular, such that every influencing or decision-making component can be easily replaced without impact on comparability of results. This provides the ability of easy model extension in future research, as well as plugging in different workload patterns, forecast metrics, and virtual instance characteristics. The following subsections describe the internal components of our model in more detail. The model’s components are formally stated and refer to the mathematical symbol definitions introduced in Table 2.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t \in \mathbb{N}, 0 \leq t \leq n$</td>
<td>$t$ denotes a certain point in time, ranging from 0 (simulation start) to $n$ (simulation end). $\Delta t = t_{i+1} - t_i$ is the simulation’s time resolution, which defaults to one minute in this work.</td>
</tr>
<tr>
<td>$R = {r_1, r_2, \ldots}$ s.t. $r = (l_i, c, k)$</td>
<td>$R$ is the set of available resources $r_i$. Each resource is defined as a 2-tuple ($l_i$, $c$), where $l_i$ refers to the resource’s load level dependent on a point in time, $c$ denotes the maximum available capacity of the resource, $k \in {\text{CPU, Memory, ...}}$ defines the type of the resource. Note, that each virtual instance’s resources are tracked individually.</td>
</tr>
<tr>
<td>$T = {\tau_1, \tau_2, \ldots}$ s.t. $\tau = (t_0, m, \delta, \rho, \varphi, s_t)$</td>
<td>$T$ is the set of all tasks $\tau_i$. Each task is defined as a 6-tuple ($t_0$, $m$, $\delta$, $\rho$, $\varphi$, $s_t$), where $t_0$ denotes the point in time the task was started, $m$ defines the task’s required time as a multiple of $\Delta t$, $\delta$ is the task’s final deadline for SLO satisfaction, $\rho \subseteq R$ are the required resources, $\varphi = [\varphi_\tau V \in \rho], 0 \leq \varphi_\tau \leq \tau, \varphi \in \mathbb{R}$ holds the amounts required per resource, $s_t \in {\text{Pending, Running, Finished, Aborted}}$ denotes the task’s current state</td>
</tr>
<tr>
<td>$V = {v_0, v_1, \ldots}$ s.t. $v = (\rho, Y, u, k)$</td>
<td>$V$ is the set of all running virtual instances $v_i$. Each virtual instance is defined as a 4-tuple ($\rho$, $Y$, $u$, $k$), where $\rho \subseteq R$ defines the provided resources, $Y \subseteq T$ are the tasks assigned to this virtual instance, $u$ is the so-called lead time of the instance, that describes the time until the instance is fully available after it has been started, $k \in {\text{Small, Medium, ...}}$ defines the type of the instance. Each instance type has a price $p_k$ assigned.</td>
</tr>
<tr>
<td>$w = [w_0, w_1, \ldots, w_n]$ s.t. $w \in \mathbb{N}$</td>
<td>$w$ is the vector of workload levels corresponding to points in time $t$. Each workload level describes the amount of new tasks that are fed into the system at the respective point in time.</td>
</tr>
</tbody>
</table>

### Performance Monitoring

For the cost-aware provisioning model, performance monitoring is required with respect to the contracted SLOs. Because the system complexity and eventual interdependence of influencing factors is already high, we focus on compliance with a single service level objective. This objective is to finish a computing task in a certain amount of time. The length of each task ranges from zero to twenty-four hours, such that a targeted maximum runtime cannot be fixed uniformly for all tasks. We therefore gear the SLO tightly to the individual runtime of each task. Hence, we can observe the dynamic system behavior with tight SLOs and loosen them step-wise to measure their impact on the cost-aware provisioning. For an informed scaling decision, the system requires performance monitoring in various scopes.
First of all, the overall system load has to be approximated. We can achieve this by having a closer look at the load of each resource provided by each virtual instance. The cost-aware provisioning aims at generating low cost solutions while maintaining the defined SLO compliance level. There are several possibilities when estimating the load of a computing system; (a) use the maximum load over all resources and instances (overestimate); (b) use the minimum load (underestimate); or (c) use the average load. Concerning a scaling decision, users may have contrary preferences, which result in different of these proposed load estimation strategies being most appropriate. Generally speaking, option (a) is best for achieving high SLO compliance over the entire runtime, since a constant overestimation leads to earlier scale-up and later scale-down actions (even though virtual instances may exist that have free capacities, they are masked by eventually only one instance that runs at very high load). On the other hand, option (b) delivers lowest possible provisioning cost to the account of SLO compliance, because sale-up actions are performed later and scale-down actions earlier. For a well-informed scaling decision based on the user’s preferences, a dynamic tradeoff between these options (a) and (b) should be used. We will introduce a parameter \( \alpha \) that defines the user’s tradeoff between high SLO compliance and low provisioning costs later. Thus, the average load over the entire system, option (c), can be applied here to achieve a load estimation that is applicable in either case of a user’s preferences. However, we perform a slight alignment towards overestimation by using the average of the most utilized resource (bottleneck resource) over all virtual instances. Next to the approximation of the overall system load, we have to be able to calculate the SLO compliance of a single task and the SLO compliance over all virtual instances.

\[
\text{System load level: } \lambda_t = \frac{\sum_{\tau \in \mathcal{T}} k_{\tau, \text{bott}}(l_{\tau, t})}{|\mathcal{V}|} \\
\text{s.t. } b = k\left(\max_{\tau \in \mathcal{T}} (l_{\tau, t})\right)^b, \text{ the bottleneck resource}
\]

The system load estimation is based on the average load of the bottleneck resource over all available virtual instances (i.e. over all resources of the same type \( b \)). Eq. 1 defines the system load \( \lambda \) depending on time \( t \), where the help symbol \( b \) denotes the type of the resource identified as bottleneck. This model applies a simplified approach of identifying a certain resource as a bottleneck. The load of each resource type (CPU, memory, etc.) is averaged over the entire cloud system (i.e. all available virtual instances). A resource is identified as the bottleneck, if its average exceeds the average of all other resource types. Even though this approach is simplified, it delivers a quite fair bottleneck estimation due to the application of load balancers that distribute new jobs to the least busy virtual instance and thus keep the overall load regulated.

The SLO compliance of a single task at a certain point in time is defined in Eq. 2. If the task deadline \( \delta \) has been met and its state can be set to ‘Finished’, the SLO has been satisfied. Otherwise we set the result to zero. This enables the system to identify all tasks that are in state ‘Finished’, sum up their \( \Lambda \) values and calculate the fraction of them that have or have not satisfied the SLO. This applies to the equations of SLO compliance for entire virtual instances defined in Eq. 3: The sum is defined to include all tasks that are assigned to the certain virtual instance \( v \) and whose state \( s_{\tau} \) is ‘Finished’. Basically, the sum counts the amount of finished tasks that have satisfied their SLO. Subsequently, the resulting value is divided by the total amount of tasks that are already finished.

\[
\text{Task SLO compliance: } \Lambda_t: (\tau, t) \rightarrow \begin{cases} 1 & \text{if } \delta_{\tau} \leq t_{\tau, t} + m_{\tau} \land s_{\tau} = \text{Finished} \\ 0 & \text{else} \end{cases} \\
\text{VM SLO compliance: } \Lambda_v: (\nu, t) = \frac{\sum_{\tau \in \mathcal{T} : s_{\tau} = \text{Finished}} \Lambda_t(\tau, t)}{|\mathcal{T}|}
\]

These definitions of load monitoring allow us to track the system performance and SLO compliance during runtime and to include them into decisions.

**Workload Forecast Model**

A stable and reliable workload forecast is essential for successful scaling policies. As already noted, we assume the daily workload to be similar setting the period length to 24 hours (cf. section “Data Generation”/Figure 7 for a seasonal pattern within 24 hours). We employ a straightforward approach of workload prediction, which requires at least some prior data to be trained. After initialization, it considers past workload values within a horizon that is larger than or equal to the period length. This ensures that seasonal data is fully eliminated for the calculation of a long-term trend. The forecast model consists of
three components shown in Eq. 4: the long-term trend, a seasonal effect, and a residual value. Both long-term and seasonal effects are predicted using past workload values, while the residual part is estimated by a Yule-Walker estimator of degree four.

Long-term trend: \[ a_t = \frac{1}{h} \sum_{i=1}^{h-1} w_t \]  
Seasonal effect: \[ e_t = \frac{1}{a} \sum_{i=t-(a \cdot (h \mod p))}^{t-p} (w_t - a_t) \]  
Residual part: \[ z_t = w_t - a_t - e_t \]  
Yule-Walker estimator: \[ \hat{z}_t = \hat{\phi}_1 z_{t-1} + \cdots + \hat{\phi}_4 z_{t-4} \]  
Forecast: \[ \hat{w}_{t+1} = \hat{z}_{t+1} + \hat{e}_{t+1} + \hat{\sigma}_{t+1} \]
s.t. \[ p \in \mathbb{N}, p > 1, \] the period length  
\[ h \in \mathbb{N}, h \geq p, \] the horizon  
\[ a \in \mathbb{N}, a = \left[ \frac{t}{p} \right], \] the amount of prior seasons  
\[ \hat{\phi}_1, \ldots, \hat{\phi}_4 \] parameters of auto-regression from Yule-Walker estimator

In detail, the long-term trend is the average over all past workload values which have the same distance to the beginning of their period as reference time \( t \) has. For example with \( p = 20 \) and \( t = 43 \), the increment variable \( i \) for calculation of the seasonal effect would be the set of values \{3, 13, 23, 33\}. The residual part is the difference between the historically calculated values and the current workload. For a forecast, only historical values are known and the residual part is estimated by application of a Yule-Walker estimator of grade 4, which is satisfactory. Thus, the forecast consists of the estimated residual part, the seasonal effect and long term trend at point in time \( t + 1 \). Using the forecast as described, we are only able to derive the expected amount of new tasks in the near future. For estimation of the future load, the system load equation can be multiplied by the fractional increase of new tasks compared to current tasks in Eq. 5.

Estimated system load: \[ \hat{\lambda}_{t+1} = \lambda_t \cdot \frac{|T| + \hat{\theta}_{t+1}}{|T|} \]

More precise and adaptive forecast techniques do actually exist, such as Fourier forecasts, machine learning techniques or wavelets, but for this case, a simple estimator with fixed period length is absolutely satisfactory.

**Scaling Decision**

The scaling decision component of the cost-aware provisioning model has the highest impact on SLO compliance. If the scaling decision fails to hit the correct point in time for a scaling operation, concerning both up-scale and down-scale, the overall SLO compliance may decrease or the overall costs may increase. In this work, we focus on four different scaling policies: static, reactive, predictive, and cost-aware predictive scaling.

\[ S = \{ \sigma_5, \sigma_R, \sigma_p, \sigma_c \} \]
\[ \sigma: (V, \theta, \hat{w}_{t+1}) \rightarrow \begin{cases} V \cup \{v \notin V\} & \text{scale up (↑)} \\ V \cap \{V_j \subset V\} & \text{scale down (↓)} \\ V & \text{no scale (=)} \end{cases} \]
\[ \theta \in (0.5 \ldots 1), \] load threshold for scaling decisions

The general definition of a scaling policy is provided in Eq. 6. The set \( S \) denotes the set of all available scaling policies, where each policy \( \sigma \) is defined as a function that maps to one out of three possible scaling actions. While the policies \( \sigma_5, \sigma_R \) operate on the set of running virtual instances and the load threshold, the policies \( \sigma_p, \sigma_c \) additionally take the predicted future workload into account. The threshold value \( \theta \) plays an important role for all scaling policies towards the achievement of a high SLO compliance while still operating at low cost: it can be adapted depending on the current SLO compliance level. For example, if the overall SLO compliance is below a certain target value, the threshold \( \theta \) is being decreased, which makes a scale-up more likely and which delays the scale-down decisions. If the system operates over the target SLO compliance level, the threshold \( \theta \) can be increased to hold the achieved level and to decrease the probability
of overprovisioning, which would implicate higher costs. Detailed descriptions and formal definitions for all scaling policies are provided in the follow-up.

The static scaling policy Eq. 7 represents a constantly overprovisioned system without any dynamic scaling at all. This is the case for most of today’s enterprise information systems and will serve as a general benchmark.

**Static policy:**

\[ \sigma_s(\cdot) \mapsto V \]  

**Eq. 7**

The reactive scaling policy (Eq. 8) is primarily based on the amount of unsatisfied task SLOs per time. If the amount of tasks with satisfied SLOs decreases from one point in time \((t - 1)\) to the next point \(t\) and the load is higher than the threshold \(\theta\) in average, the system scales up. It may only scale down if the quota of satisfied SLOs did not shrink in the last step (i.e. all tasks that have been finished at time \(t\) also have satisfied their SLOs) and if the system load will not exceed \((\theta - 0.2)\) provided that a scale-down is performed. The second condition and comparison to \((\theta - 0.2)\) is necessary, as otherwise we may run into thrashing, i.e. the system scaling one instance up and down in short periods of time. Furthermore, we ensure that no underprovision occurs. The value of 0.2 is used as the gap to avoid thrashing. Higher values lead to system overprovisioning, because scale-down actions are delayed. Lower values support incidence of thrashing and may result in scaling the system down too early (this would promote underprovisioning, since workload is volatile). For further investigations, this thrashing gap should be bound to the amount of instances that are currently available (the system state): the more instances are running, the smaller gap values are applicable, because if there are more instances residing after a scale-down, they can absorb higher amounts of tasks from the removed instance with only slightly increasing average system load.

**Reactive policy:**

\[
\sigma_R(\cdot) \mapsto \begin{cases} 
(\uparrow) & \text{if } q_t < q_{t-1} \land \lambda_t \geq \theta \\
(\downarrow) & \text{if } q_t \geq q_{t-1} \land (\lambda_t | V \cap \{v_i \in V\}) < \theta - 0.2 \\
(=) & \text{else}
\end{cases}
\]

**Eq. 8**

\[ s.t. \quad q_t = \frac{1}{f_t} \cdot \sum_{t \in T} \Lambda_t(\tau, t), \text{ the quota of tasks with satisfied SLOs} \]

\[ f_t = |\{\tau \in Y_t: s_\tau = \text{Finished } \forall v \in V\}|, \text{ the total amount of finished tasks at } t \]

For the predictive scaling policy (Eq. 9), mostly the estimated system load in the near future is decisive. Since we try to keep the system to operate at a level between \(\theta\) and \((\theta - 0.2)\), a scale up is performed whenever the estimated system load exceeds the scaling threshold. In the opposite case, the system scales down if the future system load is below the lower scaling bound \((\theta - 0.2)\) supposing the system has been scaled down. If the estimated load is within the \(\theta\) bounds, no scaling operation is required. The predictive policy does not take the current or prior SLO compliance level into account, because it is covered by the adaptation of \(\theta\).

**Predictive policy:**

\[
\sigma_p(\cdot) \mapsto \begin{cases} 
(\uparrow) & \text{if } \hat{\lambda}_{t+1} \geq \theta \\
(\downarrow) & \text{if } (\hat{\lambda}_{t+1} | V \cap \{v_i \in V\}) < \theta - 0.2 \\
(=) & \text{else}
\end{cases}
\]

**Eq. 9**

Finally, the cost-aware policy Eq. 10 operates on the same scaling conditions as the predictive policy, but additionally employs cost-aware instance selection with respect to the highest value for money of the identified bottleneck resource. That is, if memory is identified as bottleneck at a certain point in time and the policy decides to scale the system up by an additional instance, it will add an instance of the type that provides the best ratio between instance price and memory capacity. The cost-aware instance selection will also be bound to the reactive policy for comparability.

**Cost-aware policy:**

\[
\sigma_{p_c}(\cdot) \mapsto \begin{cases} 
(\uparrow) & \text{if } \hat{\lambda}_{t+1} \geq \theta, v \notin V = \min_{k \in b} \left( \frac{p_k}{c_b} \right) \\
(\downarrow) & \text{if } (\hat{\lambda}_{t+1} | V \cap \{v_i \in V\}) < \theta - 0.2 \\
(=) & \text{else}
\end{cases}
\]

**Eq. 10**


**Instance Selection**

The instance selection is the most crucial part towards cost-awareness within our scaling policy. Policies that do not apply cost-aware instance selection always replace weak by stronger instances based on the identified bottleneck resource. With cost-aware instance selection applied, the decision is made by value for money concerning the bottleneck resource. As such, the system chooses the instance type that delivers a high amount of the bottleneck resource for low costs as outlined in Eq. 11. This ensures that the system complies with the most appropriate configuration. In order to be able to choose the best additional instance, the instance selection module makes use of the straight-forward bottleneck-detection described in the previous section: the resource with the maximum load level is identified and serves as a bottleneck resource \( b = k_{\left(\max\{r_t,r\}\right)} \).

\[
\text{Instance type decision: } d = \min_{k} \left( \frac{p_k}{c_{k,b}} \right) \\
\text{s.t. } b = k_{\left(\max\{r_t,r\}\right)}, \text{ the bottleneck resource}
\]

Equation 11

Selecting the type of instance that provides the best price-capacity quota may result in overprovisioning. This excess can be avoided by incorporation of load prediction in the instance selection model. Since our model seeks to provide a general provisioning approach, the task capacity per instance type is rendered uncertain, as the resource requirements are volatile for any task. Moreover, initialization of an appropriate instance also influences the available capacity of other resources.

**Evaluation**

We have tested the presented cost-aware provisioning model using an extensive simulation. The simulation environment and all of its components are designed replaceable, such that extensibility is guaranteed. For each case of different configuration of parameters we have performed the simulation several times with varying seeds for initialization of the pseudo-random number generator. Subsequently, the results per case are averaged. The time resolution is chosen in a manner such that one simulation tick equals one real-time minute. The simulation runs were performed over 10080 ticks, which in minutes equals one complete week. The flow of our simulation is outlined in Figure 6. The factors of influence are depicted as red-shaded boxes on the right-hand side; the model’s components are represented by the white boxes. Interactions with external systems, such as receiving new tasks or performing a scaling operation on the cloud, are marked by hexagonal elements. The control flow is defined by solid arrows and flow of information is represented by dashed arrows.
Data Generation

The simulation runs can be characterized by workload patterns, task characteristics and available instance types. We aim at a realistic setup and scaling of all factors of influence. Mishra et al. (2010) identified eight different task clusters based on resource requirements focusing on the most restricted resources CPU, memory, and time. Each of these is transformed into an ordinal scale “small” (s), “medium” (m), and “large” (l) (Table 3). Computing time ranges from zero to twenty-four hours. Each task either possesses a very short or a rather long runtime. Thus, in terms of runtime the range for small tasks is zero to two hours, for large tasks it is two to twenty-four hours. Based on the task characteristics and workload traces from the Google Compute Clusters, the resulting specifications and probabilities of the different task clusters are listed in Table 4. For each cluster, there are distribution parameters that specify the mean and standard deviation of each resource in the given bounds. The SLO of each generated task is bound to its computing time requirement.

All tasks that are fed into the simulation are drawn from the identified clusters with respect to their certain probabilities. The amount of newly generated tasks depends on the current time step. We assume a typical workload pattern for weekdays as identified by Gmach et al. (2007). The typical daily pattern starts at a low utilization, increases from morning until noon, can potentially drop during lunch time, receives another increase in the early afternoon and then slowly decreases until evening (Figure 7). For simulations, we scale the workload in a manner such that there are 10 new tasks per time \( t \) at peak times.

Finally, a selection of available virtual instances is required. We stick to Amazon’s EC2 instances, but choose the basic types instead of the high-memory or high-CPU ones. For the simulation, we use the instance types “M1 Small”, “M1 Medium” and “M1 Large” listed in Table 5. The price per instance is the hourly price in USD, CPU is measured in cores (i.e. computing units in Amazon terminology), and memory is given in gigabytes. The pricing of these instances is linear concerning the cost per CPU core but non-linear regarding the memory values.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Time (in hours)</th>
<th>CPU (in cores)</th>
<th>Memory (in GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>small (s)</td>
<td>0 – 2</td>
<td>0 – 0.2</td>
<td>0 – 0.5</td>
</tr>
<tr>
<td>medium (m)</td>
<td>-</td>
<td>0.2 – 0.5</td>
<td>0.5 – 1</td>
</tr>
<tr>
<td>large (l)</td>
<td>2 – 24</td>
<td>0.5 – 4</td>
<td>1 – 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Time</th>
<th>Cores</th>
<th>Memory</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>29.85%</td>
</tr>
<tr>
<td>2</td>
<td>s</td>
<td>m</td>
<td>s/m/l</td>
<td>27.05%</td>
</tr>
<tr>
<td>3</td>
<td>s</td>
<td>l</td>
<td>s/m</td>
<td>14.2%</td>
</tr>
<tr>
<td>4</td>
<td>s</td>
<td>l</td>
<td>l</td>
<td>0.7%</td>
</tr>
<tr>
<td>5</td>
<td>l</td>
<td>s</td>
<td>s</td>
<td>17.05%</td>
</tr>
<tr>
<td>6</td>
<td>l</td>
<td>s</td>
<td>l</td>
<td>4.15%</td>
</tr>
<tr>
<td>7</td>
<td>l</td>
<td>m/l</td>
<td>s/m</td>
<td>3.65%</td>
</tr>
<tr>
<td>8</td>
<td>l</td>
<td>m/l</td>
<td>l</td>
<td>3.35%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>CPU (in cores)</th>
<th>Memory (in GB)</th>
<th>Price (in USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 Small</td>
<td>1</td>
<td>1.7</td>
<td>0.065</td>
</tr>
<tr>
<td>M1 Medium</td>
<td>2</td>
<td>3.75</td>
<td>0.130</td>
</tr>
<tr>
<td>M1 Large</td>
<td>4</td>
<td>7.5</td>
<td>0.260</td>
</tr>
</tbody>
</table>

We assume the memory resource to be restrictive and the CPU resource to be able to be overloaded. Overloading the CPU reflects heavy scheduling of tasks on a core to the expense of requiring more time to
finish for each task. In case of an overloaded CPU such as load(CPU) > 1 each task on the certain virtual instance slows down and thus is only able to proceed by \( \frac{1}{\text{load(CPU)}} \) in its computation per tick. Even though in real-world computers there are page files and the like, we do not permit memory to be overloaded. If a task is being started but is unable to satisfy its memory demand, it resides in a pending state until sufficient resources are available.

**Performance Rating**

Each simulation run indicates two different outcomes; (a) the aggregated costs for on-demand resource provisioning over the whole simulated period; and (b) the overall SLO compliance, representing the ratio between finished tasks that satisfied their SLO and the total amount of finished tasks. Since these two values improve in opposite directions, a metric for performance measurement is required that compares both outcomes and combines them into a single result that makes the performance of simulation setups intercomparable.

\[
\begin{align*}
\text{SLO compliance:} & \quad r = \frac{1}{n} \sum_{v \in V} A_v \\
\text{Total costs:} & \quad c = \sum_{t=0}^{T} \sum_{v \in V} c_{kv} \\
\text{Performance value:} & \quad \phi = (100\% - r)^\alpha \cdot c^{1-\alpha} \\
\text{s.t.} & \quad \alpha \in [0 \ldots 1], \text{the weight exponent}
\end{align*}
\]

The equations used for calculation of overall SLO compliance, total costs, and performance value are given in Eq. 12. The overall SLO compliance is the average over the SLO compliance of each virtual instance that was active at any time during simulation. The total costs result from summing up the costs for active virtual machines over all time steps \( t \). Finally, the performance value measures the difference between reached SLO compliance and the maximum possible compliance of 100 per cent as well as the costs. The weight exponent \( \alpha \) can be used to adjust the proportional valuation between low costs and high SLO compliance. If it is set to zero, the performance takes only the costs into account, if it is set to 1 only the SLO compliance level is measured. Values between these bounds represent a fair weighting towards high SLO compliance or low costs with \( \alpha = 0.5 \) being the exact balance. At the same time, the weight value removes edge cases, such as 100 per cent SLO compliance or zero costs.

**Results on Policy Performance**

The aggregated results from our extensive simulation are listed in Table 6. We have simulated various policy setups: the static policy that serves as a benchmark, and the reactive and predictive policies according to three different parameter settings. Firstly, the reactive and predictive policies are applied with a fixed scaling threshold \( \theta = 0.9 \). In the second setup, both policies were able to adapt the threshold \( \theta \) between 0.7 and 0.9 depending on the current SLO compliance level at runtime. Finally, the cost-aware instance selection was additionally applied to both policies. For each policy setup, the SLO of all simulated tasks was set to 100 per cent, 120 per cent or 150 per cent based on their certain required computation time. Each simulation case was run with a server lead time of 0, 2 and 5 minutes. The results of each simulation case are expressed in total costs for virtual instances, overall SLO compliance, and the performance value with the weight exponent set to \( \alpha = 0.3 \). For improved comparability, the performance value cells are shaded. Lighter shades relate to better performance values and vice-versa.

- The **static policy** represents an overprovisioned system, designed in a manner to cover most load spikes. In the static case, the SLO compliance is at 98.8 per cent and the total costs sum up to more than $500. Neither threshold \( \theta \), nor server lead times have an impact on the results. In our simulation results, the SLO relaxation did not affect the outcome of the static policy significantly.

- For **fixed threshold**, both reactive and predictive policies result in significantly lower total costs. However, their performance \( \phi \) is at most slightly better than using static provisioning. The improvement is weak, since scaling based on a fixed threshold has a heavy impact on the overall SLO compliance. The predictive policy is significantly better than the reactive policy.

- Using **dynamic threshold**, the policies are able to increase scaling likeliness if the target SLO is not yet reached. Thus, results increase over those with application of a fixed threshold value, because the overall SLO compliance rises. The predictive policy reaches SLO compliance values
very close to the targeted rate. The costs for each case increase over the fixed threshold setup, because policies are more likely to scale up if they operate below the targeted SLO compliance.

- The **cost-aware instance selection** mostly increases the performance values over the best-fit instance selection applied before. The SLO compliance values slightly decrease compared to prior setups. The total costs per setup improve significantly as they decrease by more than 20% on average.

The results show, that tight SLOs generally lead to lower SLO compliance rates in total. Higher SLO relaxation extends the deadline for each task and thus results in higher SLO compliance as expected. Higher lead times cause a longer time span between starting an instance and the instance becoming available to compute tasks. Against our expectations, the effect of higher lead times is only marginal concerning both costs and SLO compliance.

The **Table 6. Simulation Results** shows the values for different lead times and SLOs set by the system. The results are categorized into three groups: **Static**, **Fixed threshold**, and **Target SLO of 98%, dynamic threshold**. The table includes columns for Lead time 0, Lead time 2, and Lead time 5, with sub-columns for costs, SLO compliance, and performance values.

<table>
<thead>
<tr>
<th>Policy</th>
<th>θ</th>
<th>SLO</th>
<th>Lead time 0</th>
<th>Lead time 2</th>
<th>Lead time 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>-</td>
<td>-</td>
<td>524.21 98.8%</td>
<td>21.29</td>
<td>524.21 98.8%</td>
</tr>
<tr>
<td><strong>Fixed threshold:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reactive</td>
<td>0.9</td>
<td>100%</td>
<td>296.83 86.3%</td>
<td>29.64</td>
<td>262.33 80.7%</td>
</tr>
<tr>
<td></td>
<td>120%</td>
<td>283.9</td>
<td>89.4%</td>
<td>26.57</td>
<td>282.96 89.3%</td>
</tr>
<tr>
<td></td>
<td>150%</td>
<td>278.47</td>
<td>90.6%</td>
<td>25.31</td>
<td>283.38 90.2%</td>
</tr>
<tr>
<td>Predictive</td>
<td>0.9</td>
<td>100%</td>
<td>289.94 93.7%</td>
<td>23.10</td>
<td>286.77 95.2%</td>
</tr>
<tr>
<td></td>
<td>120%</td>
<td>289.94</td>
<td>94.7%</td>
<td>21.97</td>
<td>286.77 96.1%</td>
</tr>
<tr>
<td></td>
<td>150%</td>
<td>289.94</td>
<td>95.5%</td>
<td>20.81</td>
<td>286.77 96.6%</td>
</tr>
<tr>
<td><strong>Target SLO of 98%, dynamic threshold:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reactive</td>
<td>[0.7 – 0.9]</td>
<td>100%</td>
<td>320.97 90.7%</td>
<td>27.85</td>
<td>319.8 86.6%</td>
</tr>
<tr>
<td></td>
<td>120%</td>
<td>303.29</td>
<td>91.7%</td>
<td>25.87</td>
<td>308.49 90.5%</td>
</tr>
<tr>
<td></td>
<td>150%</td>
<td>298.79</td>
<td>92.7%</td>
<td>24.68</td>
<td>291.66 93.6%</td>
</tr>
<tr>
<td>Predictive</td>
<td>[0.7 – 0.9]</td>
<td>100%</td>
<td>336.91 97.8%</td>
<td>18.62</td>
<td>345.19 97.3%</td>
</tr>
<tr>
<td></td>
<td>120%</td>
<td>347.81</td>
<td>97.6%</td>
<td>19.61</td>
<td>327.8 97.9%</td>
</tr>
<tr>
<td></td>
<td>150%</td>
<td>338.43</td>
<td>98.0%</td>
<td>18.12</td>
<td>319.59 98.0%</td>
</tr>
<tr>
<td><strong>Cost-aware instance selection, target SLO of 98%, dynamic threshold:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reactive</td>
<td>[0.7 – 0.9]</td>
<td>100%</td>
<td>216.05 85.2%</td>
<td>24.29</td>
<td>216.75 85.4%</td>
</tr>
<tr>
<td></td>
<td>120%</td>
<td>212.05</td>
<td>85.8%</td>
<td>23.66</td>
<td>229.12 86.4%</td>
</tr>
<tr>
<td></td>
<td>150%</td>
<td>209.02</td>
<td>86.5%</td>
<td>23.08</td>
<td>205.82 87.1%</td>
</tr>
<tr>
<td>Predictive</td>
<td>[0.7 – 0.9]</td>
<td>100%</td>
<td>254.37 96.2%</td>
<td>18.12</td>
<td>313.57 97.0%</td>
</tr>
<tr>
<td></td>
<td>120%</td>
<td>293.05</td>
<td>93.5%</td>
<td>23.47</td>
<td>276.31 95.1%</td>
</tr>
<tr>
<td></td>
<td>150%</td>
<td>228</td>
<td>96.6%</td>
<td>16.26</td>
<td>235.53 96.8%</td>
</tr>
</tbody>
</table>

Aggregating the results shown in Table 6 supports the expectation and the general observation of policy tradeoffs. For visualization and clarification, costs, SLO compliance and performance values are averaged and classified into the three categories “good” (+), “fair” (−), and “bad” (−) (based on quantiles) as outlined in Table 7. Though the static policy results in a good performance value, it has very high costs at the same time. Generally speaking, reactive policies generate less costs than predictive policies but result in a lower SLO compliance and vice versa. This issue is solved by using cost-aware policies. Especially the cost-aware predictive policy results in comparatively very good values. An overview of the tradeoffs concerning the performance values for all α is provided in the following section “Managerial Implications”.

---

*Service Management and IS*

---

**Table 6. Simulation Results**
Table 7. Tradeoff Visualization for $\alpha = 0.3$

<table>
<thead>
<tr>
<th>Policy</th>
<th>Costs $c$</th>
<th>Compliance $r$</th>
<th>Performance $\phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>$- (524.21)$</td>
<td>$+ (0.988)$</td>
<td>$+ (21.29)$</td>
</tr>
<tr>
<td>Reactive</td>
<td>$+ (280.67)$</td>
<td>$- (0.863)$</td>
<td>$- (28.28)$</td>
</tr>
<tr>
<td>Predictive</td>
<td>$+ (294.28)$</td>
<td>$+ (0.955)$</td>
<td>$+ (20.95)$</td>
</tr>
<tr>
<td>Dynamic Reactive</td>
<td>$+ (305.87)$</td>
<td>$\circ (0.911)$</td>
<td>$- (26.49)$</td>
</tr>
<tr>
<td>Dynamic Predictive</td>
<td>$\circ (339.43)$</td>
<td>$+ (0.977)$</td>
<td>$+ (19.03)$</td>
</tr>
<tr>
<td>Cost-Aware Reactive</td>
<td>$+ (216.91)$</td>
<td>$- (0.861)$</td>
<td>$\circ (23.87)$</td>
</tr>
<tr>
<td>Cost-Aware Predictive</td>
<td>$+ (257.16)$</td>
<td>$+ (0.961)$</td>
<td>$+ (18.24)$</td>
</tr>
</tbody>
</table>

Managerial Implications

The performance results from our simulation can be adapted to a customer’s intention by altering the weight factor $\alpha$. The weight factor expresses the cloud customer’s preference between high SLO compliance and low total costs. If $\alpha$ is moved towards 0, the customer rates low costs higher than high SLO compliance, if $\alpha$ moves towards 1, the customer focuses on high SLO compliance and cares less about total costs. The above table only shows performance values for a fixed weight $\alpha = 0.3$. Depending on the customer’s preferences, certain policies are more or less suitable to generate the desired outcome. Figure 8 compares the results of all seven policy setups with the performance of the static scaling policy under varying $\alpha$.

The upmost graph, plotted as strong solid line, envelops the optimal policies with respect to $\alpha$ and thus marks the optimal policy for each $\alpha$. We can clearly split the optimum graph into three parts from left to right: for $\alpha \leq 0.1$, the cost-aware reactive strategy outperforms all other policies. In the interval $0.1 < \alpha < 0.4$, the cost-aware predictive strategy performs best and for $\alpha \geq 0.4$, the static policy provides the highest performance values.

The partitioning of the $\alpha$-interval clearly shows that there is no single best policy for every purpose. Based on our findings, we can derive the following two recommendations:

1. Customers who require the highest possible SLO compliance should stick to the static policy, because it is most unlikely to violate SLOs. Of course, the system needs to be properly dimensioned to even cover the highest eventual workload peak. Due to no dynamic scaling being performed, the cost-saving potential is not exhausted.
2. Customers who are satisfied with slightly lower SLO compliance but aim at lower costs should go with one of the cost-aware policies. Out of the two cost-aware policies, the predictive policy achieves...
better performance for customers who still do not want to lose focus to SLO compliance. In contrast, the reactive policy provides the best outcome for those who completely focus on minimal costs.

**Conclusion**

Cloud computing has been increasingly gaining popularity among companies for a couple of years. Low upfront costs make it an attractive alternative to in-house IT solutions. However, most cloud infrastructures still generate excessive costs, as companies tend to overprovision IT infrastructures. The reason for this over-provisioning habit lies in the buffering behavior to avoid SLA violations. In this paper, we present a cloud management framework with application of a novel cost-aware provisioning policy. Based on the crucial factors of influence (e.g. workload, task characteristics, instance characteristics), our new management concept facilitates enormous cost-saving potential.

The model we propose combines the state-of-the-art from different research threads. Previous research from the area of performance modeling and service level agreements are used to establish our performance monitoring component of our model. Our predictive policies require a reliable workload forecast for scaling decisions. Furthermore, our model is comprehensive as it accounts for all factors of influence. We validate the goodness of our model with respect to different fine-grained configurations of the factors of influence (e.g. characteristics of tasks, instances and workload) by using extensive simulations. The simulation framework we use can be adapted to any desired configuration and can thus contribute to the determination of the optimal system parameters (e.g. SLO relaxation, forecast horizon, scaling threshold). All simulation inputs stem entirely from real world data and thus guarantee a realistic simulation environment and applicability of results. The task characteristics are drawn from more than one million task signatures from the Google Compute Clusters provided by Mishra et al. (2010). Configuration and pricing of virtual instances represent the current Amazon EC2 on-demand offerings. The workload pattern used for our simulation mirrors the typical weekday patterns identified by Gmach et al. (2007).

Finally, we compare our policies by the application of a performance metric that accounts for both SLO satisfaction and operating costs in a user-weighted aggregate. We find that our novel cost-aware provisioning model delivers low-cost solutions that exhaust the cost-saving potential from statically provisioned IT infrastructures. The cost-aware provisioning is advantageous when low costs are higher valued than the maximum possible SLA compliance. Our managerial implications make the trade-off between SLA satisfaction and operating costs visible and can potentially act as a decision support for customers who can adjust their preferences accordingly.

The work presented here opens a number of avenues for future research. First, we will extend our simulation by confronting our cloud management system with large load traces, such as delivered by TPC (2013). Additionally, we aim to include a consumer-side risk management approach into our proposed provisioning model. The risk-awareness of cloud customers is expected to have a remarkable impact on provisioning policies and thus also influences cost-savings and SLO compliance. Another aspect that is considered for future work is the competition between tasks based on the valuation of their respective owners. Currently, all tasks are weighted and treated equally in our simulations, as we apply a straightforward earliest-deadline-first scheduling. Based on a task’s resource requirements or the respective owner, it may be more valuable and could be prioritized over other tasks. Alternatively, an auction-based approach is imaginable where task owners can bid for the prioritization of their tasks. For analysis of the reliability and stability of our provisioning model and the applied policies, the system can be exposed to unpredictable load spikes. The impulse-response behavior will have significant impact on total costs and SLO compliance.
References


Toronto City Summit Alliance 2010. Greening Greater Toronto.

