# Impact of User Patience on Auto-Scaling Resource Capacity for Cloud Services

Marcos Dias de Assuncao, Carlos Cardonha, Marco Netto, Renato Cunha

# Impact of User Patience on Auto-Scaling Resource Capacity for Cloud Services

Marcos Dias de Assunção[a,*], Carlos H. Cardonha[b], Marco A. S. Netto[b], Renato L. F. Cunha[b]

[a] *Inria, ENS de Lyon, France*
[b] *IBM Research, Brazil*

**Abstract**

An important feature of most cloud computing solutions is auto-scaling, an operation that enables dynamic changes on resource capacity. Auto-scaling algorithms generally take into account aspects such as system load and response time to determine when and by how much a resource pool capacity should be extended or shrunk. In this article, we propose a scheduling algorithm and auto-scaling triggering strategies that explore user patience, a metric that estimates the perception end-users have from the Quality of Service (QoS) delivered by a service provider based on the ratio between expected and actual response times for each request. The proposed strategies help reduce costs with resource allocation while maintaining perceived QoS at adequate levels. Results show reductions on resource-hour consumption by up to approximately 9% compared to traditional approaches.

*Keywords:* Cloud computing, auto-scaling, resource management, scheduling

## 1. Introduction

Cloud computing has become a popular model for hosting enterprise and backend systems that provide services to end-users via Web browsers and mobile devices [1, 2]. In this context, a typical scenario often comprises a provider of computing and storage infrastructure, generally termed as Infrastructure as a Service (IaaS) or simply a *cloud provider*; a *service provider* who offers a web-based service that can be deployed on Virtual Machines (VMs) hosted on the cloud; and the *clients*, or *end-users*, of such a service.

In this work, we investigate challenges faced by the service providers who compose their resource pools by allocating machines from cloud providers. Meeting end-users expectations is crucial for the service provider's business, and in the context of Web applications, these expectations typically refer to short requests response times. Simultaneously, there are costs associated with resource allocation, so resource pools with low utilisation are economically undesired. Moreover, clients demands are uncertain and fluctuate over time, so the problem of resource allocation faced by service providers is clearly non-trivial.

Elasticity, a selling point of cloud solutions, enables service providers to modify the size of resource pools in near real-time via auto-scaling strategies, allowing hence for reactions to fluctuations on clients' demand. Current strategies typically monitor and predict values of target system metrics, such as response time and utilisation level of relevant resources (*e.g.*, CPU, memory, and network bandwidth), and employ rule-based systems to trigger auto-scaling operations whenever predefined thresholds are violated. The parameters employed by these strategies do not allow them to explore *heterogeneity of expected response times* and *of tolerance to delays* that end-users have towards service providers. In combination with actual response times, these two elements define the perception end-users have from a service QoS, which we will refer to as *user patience*.

Previous work in other domains has shown that end-users can present heterogeneous patience levels depending on their context [3]. Moreover, in a society where human attention is increasingly becoming scarce, and where users perform multiple concurrent activities [4][1] and use multiple devices [5][2], response time might not be the sole element defining the perceptions end-users have from a service's QoS. Our previous work investigated how application instrumentation [6, 7, 8], and end-user context and profiling [9] could be used in the

*Corresponding author: assuncao@acm.org

[1] Ofcom reports that up to 53% of UK adults media multi-task while watching TV.

[2] Accenture's study shows users expect other devices to augment content shown on TV.

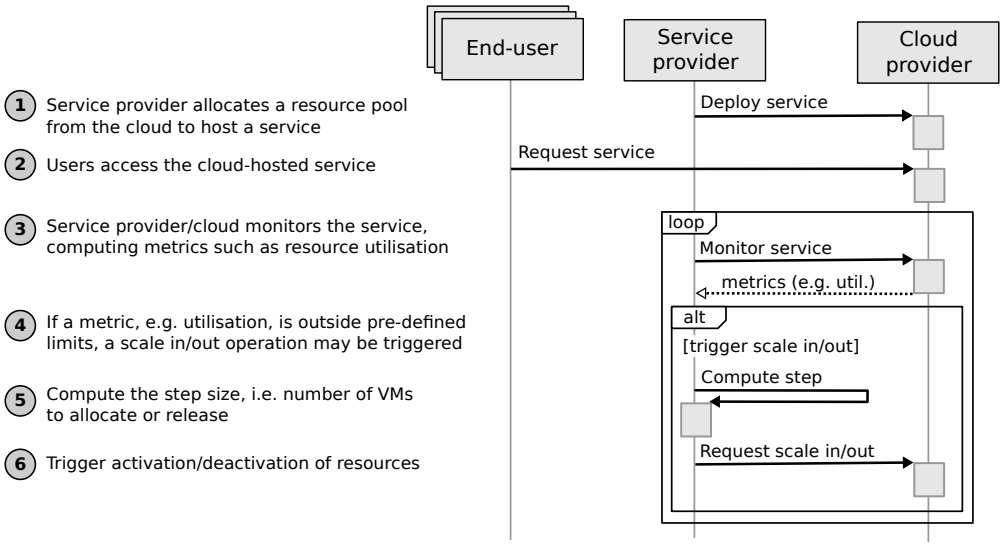


Figure 1: Service provision model and auto-scaling scenario considered in this work.

collection of honest signals determining how clients interact with a service provider and what their levels of patience are when making requests.

The present work investigates how patience can be explored by auto-scaling strategies. Compared to our previous work [8], in addition to considering provisioning time, this article makes the following contributions:

- A scheduling algorithm and a patience-based auto-scaling triggering strategy for a service provider to minimise the number of resources allocated from a cloud provider, applicable to scenarios where the maximum number of available resources is unbounded (§ 3);
- Experiments with bounded and unbounded numbers of resources available to service providers that show reductions of up to nearly 9% on resource-hour consumption compared to traditional auto-scaling strategies (§ 4).

## 2. Problem Description

In this section we describe the scenarios investigated in this work, present the assumptions regarding capacity limitations on the pool of resources offered by the cloud provider, and explain the elements of client behaviour which are taken into account by our algorithms.

Figure 1 depicts the service hosting model and the main steps of auto-scaling operations. A service provider allocates a pool of VMs from a cloud provider to deploy its service, which is then accessed by end-users. The service provider and/or cloud provider periodically monitors the status of the pool, thus computing metrics such as resource utilisation. A metric lying outside certain lower and upper limits may trigger a scale in/out operation. A step size — the number of VMs to be allocated or released — is computed before a change to the capacity of the resource pool is requested.

Resources are pairwise indistinguishable, that is, they are VMs which have the same cost and performance characteristics. We also consider provisioning time; namely, after triggering the activation of resources, a service provider must wait for non-negligible time to use them. Finally, the scenarios either have a maximum number of resources that can be allocated (*bounded*) or do not have limitations of this nature (*unbounded*); these two scenarios are considered because they pose different levels of stress on resource utilisation.

We take into account that resources are paid only for the periods in which they were allocated in a per-minute billing model. This assumption is not unrealistic, as providers such as Microsoft Azure offer this type of service. Consequently, the strategies proposed in this article may allocate and deallocate a given resource several times within one hour in order to improve utilisation.

This work also considers short tasks whose execution times are all equal and in the order of a few seconds, reflecting hence scenarios typically faced by providers of web services. Nevertheless, the concepts and results can, without any loss of generality, be reproduced in scenarios with either shorter or longer tasks.

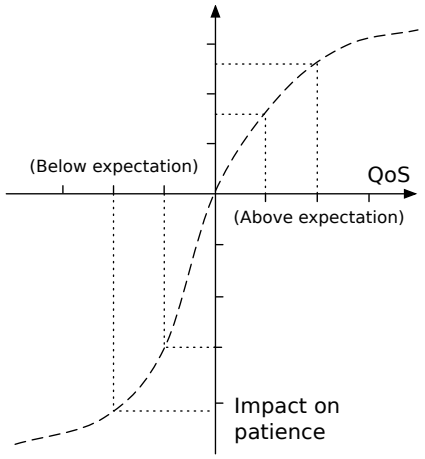


Figure 2: Prospect theory modelling variations of user patience as a function of QoS [10]; where QoS can represent expectations on, for instance, request response time and jitter.

We assume that expectations on response time may vary across individuals, a realistic assumption in certain practical settings; for example, whereas end-users expect to get results from Web searches in a couple of seconds at maximum, clients performing large-scale graph mining operations, which can take from minutes to hours, tend to be more tolerant.

Differences between expected and actual response times for submitted requests have an impact on users' patience (either positive or negative) whose weight decreases over time. More precisely, changes on the patience level of an end-user take place after the execution of each request and are described by a function applied to the ratio between expected and actual response times. Based on Prospect Theory [10], we assume that the negative impact of delays on users' patience exceeds the benefits of fast responses (see Figure 2).

The service provider periodically evaluates the system utilisation and/or user patience and decides on whether the capacity of its resource pool should be expanded (shrunk) by requesting (releasing) resources from (to) the cloud. The questions we therefore seek to address are: *(i) how to determine critical times when auto-scaling is necessary or avoidable by exploiting information on users' patience; (ii) how to determine the number of resources to be allocated or released; and (iii) how strategies behave under scenarios with and without restrictions on the maximum number of resources that can be allocated.*

## 3. Algorithms for Auto-Scaling and Scheduling

This section starts with a mathematical description of the challenges a service provider faces to set its resource pool capacity. Based on this formalisation, we introduce the main contributions of this work: auto-scaling strategies and a scheduling algorithm based on user patience. Table 1 summarises the employed notation.

### *3.1. Mathematical Description*

We consider a discrete-time state-space model where time-related values are integers in $\mathcal{T} \subseteq \mathbb{N}$ representing seconds. There is a set $\mathcal{U}$ of end-users or clients and a sequence $\mathcal{J}$ of incoming requests to a service; the end-user who submitted request $j \in \mathcal{J}$ is denoted by $u(j)$.

The service provider allocates up to $m$ resources from the cloud provider in order to host its service and handle end-users' requests, and its resource pool capacity at time $t \in \mathcal{T}$ is denoted by $m_t$ for each $t \in \mathcal{T}$; recall that, with auto-scaling, $m_t$ may be different from $m_{t'}$ for $t \neq t'$. We consider that $m_t \geq b$, where $b \in \mathbb{N}$ is defined *a priori* by the service provider; setting such a lower bound is a common practice in the industry to prevent resource shortage under low, yet bursty load. Moreover, $w_t$ represents the number of resources in use to handle requests at time $t$; note that $w_t < m_t$ indicates that $m_t - w_t$ active resources are not being used by the service provider.

A request occupies one resource for $\rho$ seconds in order to be processed, $\rho \in \mathbb{N}$, and that request decomposition (*i.e.*, execution of sub-parts in different machines or times) and preemption are forbidden. Due to the amount of time spent with queueing and scheduling operations, though, response times are almost always higher than $\rho$. Based on previous interactions with the service provider, the response time that end-user $u$ expects for each submitted request is given by $\rho \times \beta_u$, where $\beta_u \in \mathbb{R}$ is a multiplicative factor larger or equal than 1; we assume that $\beta_u$ does not change over time. The actual response time for request $j$ is denoted by $r_j$.

End-user $u$'s patience level at instant $t$ is denoted by $\phi_{t,u} \in \mathbb{R}^+$. If $\phi_{t,u}$ is below a certain threshold $\tau_u \in [0, 1)$, we say that $u$ is unhappy. Based on the ideas introduced by Prospect Theory — where the sense of losing an opportunity transcends the feeling of gaining it — $\phi_{t,u}$ behaves as follows. First, for every request $j$, $u(j)$'s reaction to response time $r_j$ is given by

$$x_j = \frac{\rho \times \beta_{u(j)}}{r_j},$$

where $x_j > 1$ indicates that $r_j$ was smaller than expected, thus making $u(j)$ positively "surprised"; $x_j < 1$

accounts for cases where the $r_j$ was greater, making $u(j)$ "disappointed"; and $x_j = 1$ when equality between expected and actual response times hold. Considering the different user reactions suggested by Prospect Theory, we assume in this work that patience level $\phi_{t,u}$ changes according to $x_j$ as follows:

$$\phi_{t,u} = \begin{cases} \alpha_{1,u}\phi_{t-1,u} + (1-\alpha_{1,u})x_j, \text{if } x_j > 1; \\ \alpha_{2,u}\phi_{t-1,u} + (1-\alpha_{2,u})x_j, \text{otherwise,} \end{cases}$$

where $0 \leq \alpha_{1,u}, \alpha_{2,u} \leq 1$. Exponential smoothing is used in both cases to update the patience level. In order to make the immediate impact of slower response times higher than that of faster responses, we have $\alpha_{1,u} > \alpha_{2,u}$; moreover, different end-users may have different values of $\alpha_{1,u}$ and $\alpha_{2,u}$. In summary, each end-user $u$ is described by a tuple $(\beta_u, \alpha_{1,u}, \alpha_{2,u}, \tau_u, \phi_{0,u})$.

### *3.2. Auto-Scaling and Scheduling Strategies*

In order to set an auto-scaling strategy, a service provider needs to define *when* an scaling operation should take place and to select adequate *step sizes*, which are the number of resources by which the resource pool capacity should extend or shrink.

Typically, resource utilisation is used to trigger scale-in and scale-out operations. Resource utilisation at time $t$, denoted by $\upsilon_t$, is the ratio between the number $w_t$ of resource hours effectively used to handle requests and the number $m_t$ of resource hours allocated from the cloud (*i.e.*, idle resources are not considered). The service provider sets parameters $H$ and $L$, $0 \leq L \leq H \leq 1$, indicating utilisation thresholds according to which resources are activated and deactivated, respectively. Observe that, irrespective of scale-in decisions, though, resource pool capacity will never be inferior to $b$.

Ideally, resource pool capacity is modified before utilisation reaches undesired levels. For this reason, service providers typically rely on predictions of resource utilisation. In this work, this estimation is based on the measurements performed over the past $i$ measurement intervals for some $i \in \mathbb{N}$. Namely, after measuring $\upsilon_t$ at time $t$, weighted exponential smoothing is used to predict the utilisation for step $t+1$. If the past $v \leq i$ measurements (*i.e.*, $\upsilon_{t-v}, \upsilon_{t-v+1}, \ldots, \upsilon_t$) and the forecast utilisation are below (above) the lower (upper) threshold $L$ ($H$), the scheduler triggers a scale-in (scale-out) operation.

We propose the following auto-scaling strategies:

- *Utilisation Trigger (UT):* a strategy that uses system utilisation and employs a formula to identify an upper bound on step sizes for activating/releasing resource capacity;

Table 1: Summary of the notation used in this article.

| Symbol | Description |
|---|---|
| $\mathcal{T}$ | Discrete-time state-space model |
| $\mathcal{U}$ | Set of end-users of the service |
| $\mathcal{J}$ | Sequence of requests made by end-users |
| $u(j)$ | End-user who submitted request $j$ |
| $m$ | Service provider's maximum resource pool capacity |
| $m_t$ | Service provider's resource pool capacity at time $t$ |
| $b$ | Service provider's minimum resource pool capacity, set *a priori* |
| $w_t$ | Number of resources in use at time $t$ |
| $\rho$ | Processing time of a request |
| $\beta_u$ | Factor that determines the expected response time end-user $u$ has after submitting requests |
| $r_j$ | Actual response time for request $j$ |
| $\phi_{t,u}$ | End-user $u$'s patience level at time $t$ |
| $x_j$ | End-user $u(j)$'s reaction to response time $r_j$ |
| $\alpha_{1,u}$ and $\alpha_{1,u}$ | Parameters that define how the patience level of end-user $u$ varies over time |
| $\upsilon_t$ | Resource utilisation at time $t$ |
| $L$ | Lower bound for resource utilisation |
| $H$ | Upper bound for resource utilisation |
| $i$ | Number of intervals for predicting resource utilisation |
| $\lambda$ | Percentage of $m$ which equals step size |
| $s$ | Resource step size (*i.e.*, number of resources to activate/deactivate in an auto-scaling operation) |
| $s_t$ | Resource step size at time $t$ |
| $\gamma$ | It determines the aggressiveness of auto-scaling strategies when computing the step size $s_t$ |
| $k_j$ | Number of responses with bad QoS $u(j)$ will accept before having patience limit exceeded |
| $\hat{K}_t$ | Average value of $k_j$ for all the requests in the service (either enqueued or in execution) |
| $\tau_{u(j)}$ | Maximum response time a user $u$ is willing to endure after receiving a response for $j$ |
| $a_t$ | Number of end-users accessing the service at time $t$ |
| $L_t$ | Lower bound derived for step sizes in scenarios where $m = \infty$ |
| $U_t$ | Upper bound derived for step sizes in scenarios where $m = \infty$ |
| $l_t$ | System load at time $t$ according to workloads |

- *Unbounded-resource-pool Utilisation Trigger (UUT)*: uses system utilisation and employs a formula to determine an upper bound and a lower bound on step sizes for activating/releasing resource pool capacity, based on previous work [11];
- *Utilisation and Patience Trigger (UPT):* extends the first strategy by using user patience to compute dynamically a factor for multiplying the step sizes suggested by the formula;
- *Unbounded-resource-pool Utilisation and Patience Trigger (UUPT):* extends the second strategy by using user patience in order to choose dynamically a value from the step size interval defined by the formula.

The rest of this section details the main principles of the auto-scaling strategies and the scheduling algorithm.

*3.2.1. Utilisation-based Auto-Scaling*

Let $\lambda \in [0, 1]$ be the percentage of $m$ by which the resource pool extends or shrinks in each auto-scaling operation, *i.e.*, let $s = \lfloor \lambda \times m \rfloor$. When setting utilisation thresholds $L$ and $H$ and step size $s$, service providers want to prevent the occurrence of opposite auto-scaling operations in consecutive steps. Such events take place in two situations, which can be avoided if we assume that $w_t = w_{t+1}$ as we discuss below.

First, we have a scale-out operation taking place at time $t$ and a scale-in operation taking place at time $t + 1$ if $\frac{w_t}{m_t} > H \implies w_t > Hm_t$, $\frac{w_{t+1}}{m_t+\lambda\times m} < L \implies w_{t+1} < L \times (m_t + \lambda \times m)$, and $w_t \leq w_{t+1}$ hold simultaneously. This can be avoided if the following inequality holds:

$$\begin{aligned} Hm_t &> L \times (m_t + \lambda \times m) \implies \\ \frac{H}{L} &> \frac{m_t + \lambda \times m}{m_t} \implies \\ \frac{H}{L} &> 1 + \frac{\lambda \times m}{m_t}. \end{aligned}$$

A similar analysis indicates how to avoid scale-in operations at time $t$ and scale-out operations at $t + 1$ for any $t \in \mathcal{T}$ whenever $w_t \geq w_{t+1}$. Namely, we want to avoid $\frac{w_t}{m_t} < L \implies w_t < Lm_t$ and $\frac{w_t}{m_t-\lambda\times m} > H \implies w_t > H \times (m_t - \lambda \times m)$ holding simultaneously; for this, we need

$$\frac{H}{L} > 1 + \frac{\lambda \times m}{m_t - \lambda \times m}.$$

Moreover, we have that

$$\begin{aligned} \frac{\lambda \times m}{m_t - \lambda \times m} &> \frac{\lambda \times m}{m_t} \implies \\ m_t &> m_t - \lambda \times m \implies \\ \lambda \times m &> 0, \end{aligned}$$

so both inequalities are satisfied if $w_t = w_{t+1}$ and $\frac{H}{L} \geq 1 + \frac{\lambda\times m}{m_t-\lambda\times m}$. As the service provider will keep at least $b$ machines active, we have that $m_t - \lambda \times m \geq b$. Therefore, the parameters have to obey the relation below:

$$\frac{H}{L} > 1 + \frac{\lambda \times m}{b}.$$

From the above, one derives an upper bound for the step size $s = \lambda \times m$. This value is derived from borderline scenarios, though, so it yields step sizes that are too large in most cases. Therefore, service providers can choose a multiplicative factor $\gamma \in [0, 1]$ indicating the strategy's "aggressiveness"; namely, given $\gamma$, the step size will be $s \times (1 - \gamma)$ and $s \times \gamma$ for scale-in and scale-out operations, respectively.

Utilisation Trigger (UT) is an implementation of the ideas discussed above; namely, in order to use UT, a service provider only needs to set utilisation thresholds $L$ and $H$ and factor $\gamma$. Observe that this strategy is based solely on utilisation and assumes that $m$ is finite.

When a service provider is not subject to upper bounds on the number of resources it can allocate (*i.e.*, $m = \infty$), the formulae employed to derived upper bounds on step sizes used by UT cannot be applied. Under these circumstances, we employ Unbounded-resource-pool Utilisation Trigger (UUT), a strategy introduced in our previous work [11] in which the upper bound on $s$ for scale-out operations is given by

$$s \leq m_t \frac{\upsilon_t - L}{L}$$

and the lower bound by

$$s \geq m_t \frac{\upsilon_t - U}{U},$$

whereas, for scale-in operations, the upper bound is

$$s \leq m_t \frac{L - \upsilon_t}{L}$$

and the lower bound is

$$s \geq m_t \frac{U - \upsilon_t}{U}.$$

Let $L_t$ and $U_t$ be the lower and upper bounds derived from the inequalities above at time $t$, respectively. In addition to $L_t$ and $U_t$, UUT is also given an aggressiveness factor $\gamma \in [0, 1]$. Given these parameters, for scale-in operations the value of $s_t$ is

$$s_t = (1 - \gamma) \times L_t + \gamma \times U_t,$$

whereas for scale-out operations it is

$$s_t = \gamma \times L_t + (1 - \gamma) \times U_t.$$

More details are provided in previous work [11].

*3.2.2. Patience-based Scheduling and Auto-Scaling*

Lowest Patience First (LPF) is a scheduling algorithm that orders arriving requests based on end-users' patience level as follows. If, at instant $t$, end-user $u$ submits request $j$ and $x_j < 1$, we have

$$\phi_{t+1,u} = \alpha_1 \phi_{t,u} + (1 - \alpha_1) x_j \geq \alpha_1 \phi_{t,u}.$$

Since

$$\lim_{r_j \to \infty} \frac{\rho \times \beta_u}{r_j} = 0,$$

$\phi_{t+1,u}$ becomes closer to $\alpha_1 \phi_{t,u}$ as $r_j \to \infty$. Therefore $\alpha_1 \phi_{t,u}$ is a lower bound for $\phi_{t+1,u}$, and the accuracy of this approximation grows as system load (and, consequently, response times) gets larger. Using this bound, we can estimate the minimum value $k_j \in \mathbb{N}$ for which $\phi_{t+k_j,u}$ becomes smaller than $\tau_u$ as follows:

$$\begin{aligned} \alpha_1^{k_j} \phi_{t,u} &\geq \tau_u \implies \\ \alpha_1^{k_j} &\geq \frac{\tau_u}{\phi_{t,u}} \implies \\ k_j &\geq \log_{\alpha_1}\left(\frac{\tau_u}{\phi_{t,u}}\right); \end{aligned}$$

namely, value $k_j$ indicates the number of submissions for which $u(j)$ will tolerate a bad QoS before having $\tau_{u(j)}$ surpassed.

LPF sorts the requests in the scheduling queue according to $k_j$ (smaller values first). In order to avoid starvation, $k_j$ is set to zero if request $j$ is waiting for a period of time that is at least twice as large as the current average response time.

Under UT and UUT $\gamma$ is fixed, so the service provider is forced to have the same behaviour (aggressive or conservative) in each auto-scaling operation. UPT and UUPT leverage UT and UUT, respectively, by employing user patience to set dynamically the value of $\gamma$.

Let $\hat{K}_t$ denote the average value of $k_j$ for all the requests in the service provider (either enqueued or in execution) at time-step $t$. Whenever the system decides that an auto-scaling operation should be performed, a sequence $S$ containing the last $k$ averages $\hat{K}_t$ is created and has some percentage of its largest and smallest values removed. From the remaining elements of $S$, the system takes the largest value $K$. Finally, the value of $\gamma$ will be $1 - \frac{\hat{K}_t}{K}$ and $\frac{\hat{K}_t}{K}$ for scale-out and scale-in operations, respectively.

Essentially, UPT and UUPT adapt the system behaviour according to users' current average-patience level. Namely, if $\hat{K}_t$ is high, a more aggressive policy is acceptable. Conversely, if $\hat{K}_t$ is low, a conservative approach could support the improvement of QoS.

## 4. Evaluation

We conducted extensive computational experiments in order to investigate the benefits and drawbacks of the algorithms presented in Section 3. We modelled scenarios where the maximum number of resources can be either bounded or unbounded, and evaluated the following combinations of task scheduling and auto-scaling triggering strategies:

- **FIFO+UT** or **FIFO+UUT**: First-In, First-Out (FIFO) scheduling with auto-scaling considering only resource utilisation.
- **LPF+UT** or **LPF+UUT**: LPF scheduling with auto-scaling considering only resource utilisation.
- **LPF+UPT** or **LPF+UUPT**: LPF scheduling with auto-scaling considering resource utilisation, and end-users' patience to define the resource step size of scale-out and scale-in operations.

Our comparisons employed the following metrics:

- **Percentage of dissatisfactions**: percentage of requests $j$ whose response time $r_j$ led or kept end-user $u(j)$'s patience level below threshold $\tau_{u(j)}$;
- **Percentage of QoS violations**: percentage of requests whose response times were longer than the amount of time expected by their end-users;
- **Allocated resource-time:** resource capacity—in machine-seconds—allocated to serve user requests.
- **Aggregate request slowdown:** consists of the accumulated sum of all delays suffered by requests involved in QoS violations as is therefore given by

$$\sum_{j \in \mathcal{J}} \max(r_j - \beta_{u(j)} \times \rho, 0).$$

The rest of this section describes the experimental setup and discusses the obtained results.

### *4.1. Experimental Setup*

A built-in-house discrete-event simulator was used to evaluate the performance of the proposed strategies. We crafted two types of workloads with variable hourly-load over a 24-hour period. The rationale behind these workloads, depicted in Figure 3, is as follows:

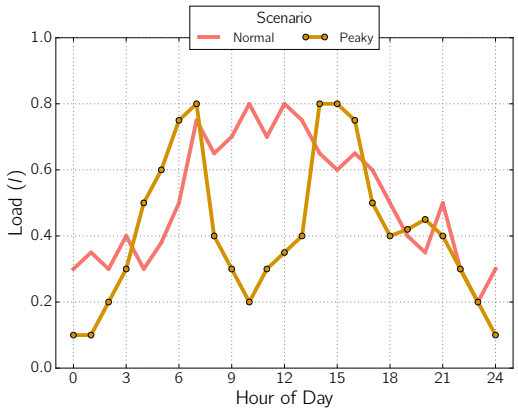


Figure 3: The workloads considered in this work, where the hourly load $l$ is employed to determine the number of end-users accessing the service.

- *Normal day*: consists of small peaks of utilisation during the start, middle, and towards the end of working hours. Outside these intervals, but still during working hours, the workload remains around the peak values, whereas outside the working hours the load decreases significantly.
- *Peaky day*: consists of tipping workload peaks, a scenario in corporations where requests need to be executed only at a few moments of the day.

The workloads are employed to determine the number of users accessing a service at a given time $t$. Every 10 seconds, the number $a_t$ of end-users accessing the service is adjusted to

$$a_t = |\mathcal{U}| \times (l_t + \eta),$$

where $|\mathcal{U}|$ denotes the maximum number of users who can be accessing the service at any given moment, $0 \geq l_t \leq 1$ is the load at time $t$ according to the crafted workloads (representing hence a fraction of $|\mathcal{U}|$), and $\eta$ is uniformly distributed over $-0.05$ to 0.05 in order to add a random variation of 5%; the sum $l_t + \eta$ is always set to the closest value in the interval [0,1]. For instance, if at time $t$, $|\mathcal{U}| = 100$, $l_t = 0.7$ and $\eta = 0.05$, the number of users accessing the system and making requests is $a_t = 75$. For intermediate values of $t$, $a_t = a_{t-1}$. Unless otherwise noted, the experiments described here use $|\mathcal{U}| = 1,000$.

Requests have processing time $\rho$ of 10 seconds, whereas end-user's expected response time for request $j$ is $\rho$ multiplied by $\beta_{u(j)} \geq 1$; *e.g.*, if $\beta_{u(j)} = 1.3$, the expected response time is 13 seconds. Request inter-arrival times, or end-users "think time"—amount of time in seconds separating the arrival of a response for request $j$ and the submission of a new request by active end-user $u(j)$—is drawn uniformly from 0 to 100 for each request submission, that is, it is not constant for any end-user. The values of $\alpha_1$ and $\alpha_2$ are given by 0.2 and 0.1 multiplied by numbers drawn uniformly for each end-user from $[0.9, 1.1]$, respectively. Finally, each $\tau_u$ and $\phi_{u,0}$ are random numbers drawn uniformly from $[0.45, 0.55]$ and $[0.8, 1.0]$, respectively. These values are used by the algorithms employed by the service provider, so it can be assumed that they have been obtained or estimated based on historical data associated with each end-user.

We tested bounded scenarios in which maximum resource pool capacity was an even value between 162 to 180; in these cases, the initial capacity $m_0$ and the minimum capacity $b$ are set to 25% of the maximum. For unbounded scenarios, initial/minimum capacity is set to 45 resources. The rationale behind these values is simple. Whereas it is a common practice in the industry to set a minimum resource capacity for a service, we do not want this minimum capacity to be large enough to eclipse the impact of auto-scaling decisions.

The lower and upper target thresholds for resource utilisation (*i.e.*, $L$ and $H$) are 40% and 70% respectively, which are default utilisation levels used by existing auto-scaling services, such as Amazon Elastic Compute Cloud (EC2). Resource utilisation levels are measured every 60 seconds. The value of $\gamma$ is set to 0.25 for strategies UT and UUT under bounded scenarios and to 0.5 under unbounded scenarios, respectively.

Provisioning time in each scale-out operation is measured in seconds and is drawn from a Normal distribution with mean 180 and standard deviation 15; sampling is performed in an aggregate way, that is, a single value is used by all resources involved in the same operation. These values were chosen based on existing work that identified that the mean time required to provision lean VM instances in certain clouds is often over 100 seconds [12]. Moreover, the fact that in a group of VMs there are certain VMs that can boot slower and that the provision of VMs with Web server software stack takes longer as demonstrated in the existing work, we added extra 80 seconds. We performed experiments with higher mean values (up to 10 minutes) and obtained similar conclusions for the proposed techniques (therefore, due to space constraints, we report only on results involving provisioning time variable to $180 \pm 15$). Deallocation time was not considered in this work because it is typically much shorter than provisioning time and its effects can only be observed in scenarios where utilisation changes abruptly and quickly, which is not

the case of the workloads considered in this work.

Finally, we simulated the equivalent of 4 days of activities of a service provider in order to reduce biases that single executions could have brought to the final results.

### *4.2. Result Analysis*

**Bounded number of resources.** We start discussing the results of our experiments with scenarios where the maximum resource pool capacity is bounded. These results are summarised in Figures 4, 5, 6, 7, 8, and 9.

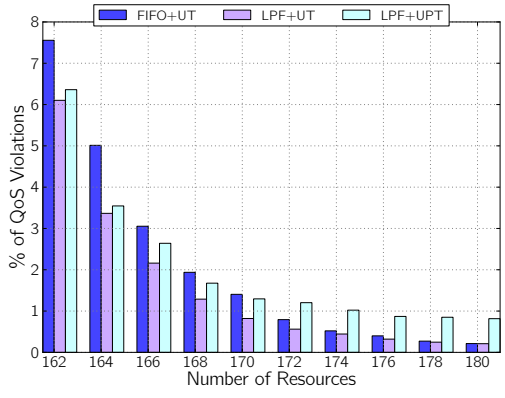


Figure 4: **Bounded:** Percentage of QoS violations for FIFO+UT, LPF+UT, LPF+UPT on the Normal workload.

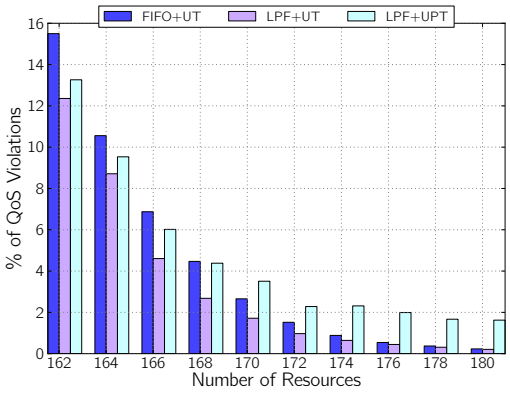


Figure 5: **Bounded:** Percentage of QoS violations for FIFO+UT, LPF+UT, LPF+UPT on the Peaky workload.

Figures 4 and 5 present the percentage of requests whose response times were larger than expected. In general, LPF+UPT is superior and to FIFO+UT in scenarios with 162-170 and worse than FIFO+UT with 172-180 resources, respectively, whereas LPF+UT is

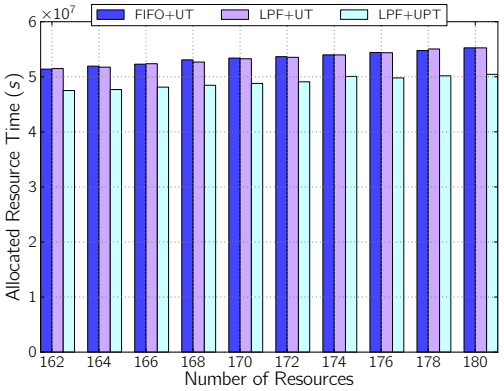


Figure 6: **Bounded:** Comparison of resource-time usage for FIFO+UT, LPF+UT, LPF+UPT on the Normal workload.

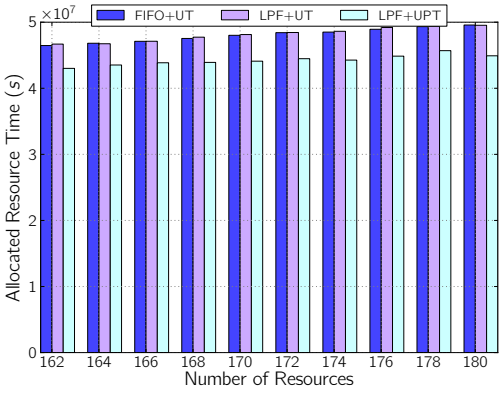


Figure 7: **Bounded:** Comparison of resource-time usage for FIFO+UT, LPF+UT, LPF+UPT on the Peaky workload.

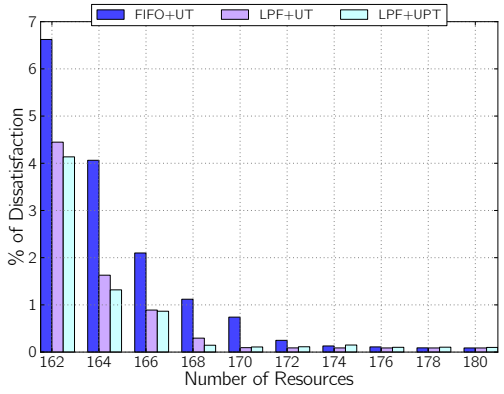


Figure 8: **Bounded:** Percentage of user dissatisfaction for FIFO+UT, LPF+UT, LPF+UPT on the Normal workload.

better than both in all scenarios with respect to this metric. Poor results from FIFO+UT are not surprising, as

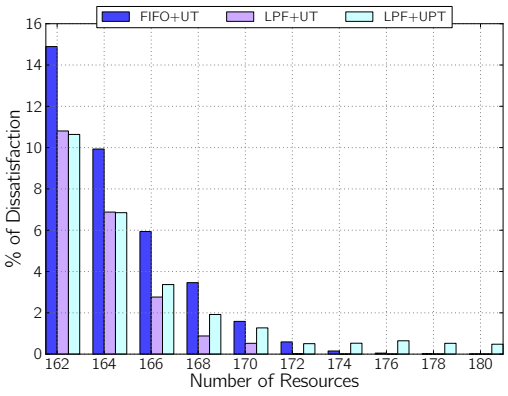


Figure 9: **Bounded:** Percentage of user dissatisfaction for FIFO+UT, LPF+UT, LPF+UPT on the Peaky workload.

it is completely oblivious to user patience. The comparison between LPF+UT and LPF+UPT is less straightforward; LPF+UT leads to fewer QoS violations than LPF+UPT, but Figures 6 and 7 show that the later compensates this with significant reductions on allocated resource time (approximately 9%).

Additionally, results of LPF+UPT on percentage of QoS violations and on percentage of dissatisfactions for scenarios with 170-180 resources show the effects of making $\gamma$ dynamic. Namely, in scenarios where the maximum number of resources is large enough to satisfy demand, users' patience level are more likely to stay high. Whenever auto-scaling operations have to be performed, the algorithm assigns "aggressive" values to $\gamma$ that enforce the allocation (deallocation) of a very small (large) number of machines in scale-out (scale-in) operations. As a result, response time for a certain number of requests may be larger than expected in situations where such delays could have been avoided. However, the results presented in Figures 8 and 9 for 172-180 resources suggest that it is possible to provide insufficient QoS for approximately 1-2% of all requests without perceptible damages to user patience in such scenarios (as illustrated by LPF+UPT in Figures 4 and 5).

Figures 8 and 9 show a clear superiority (*i.e.*, lower percentages) of LPF+UT and a clear inferiority of FIFO+UT with respect to end-user dissatisfaction. The latter phenomenon can be again explained by the fact that FIFO does not take patience into account. Moreover, differences between LPF+UT and LPF+UPT are again counterbalanced by differences on the volume of resource-time usage for each configuration. The results of scenarios with 174-180 resources are inconclusive, as the percentage of dissatisfied users in these cases is insignificant; because queues are almost never formed in these situations, the scheduling strategies basically deliver the same solutions and auto-scaling algorithms have enough time to adapt the resources' pool.

Finally, the results show that keeping user patience in satisfactory levels is more challenging for the Peaky workload. The overall resources demand on both scenarios are different, but whereas differences on the volume of allocated resources are relatively low (approximately 10%), the percentages of QoS violations and dissatisfactions differ by a factor of 2. Peaky workloads model situations where resource demand increases abruptly, and under these circumstances, the number of QoS violations tend do increase considerably. Therefore, it is natural to expect lower user satisfaction levels in such scenarios.

**Unbounded number of resources.** We present now results involving the simulation of scenarios where service providers could provision as many resources as they wished. These results are summarised in Figures 10, 11, 12.

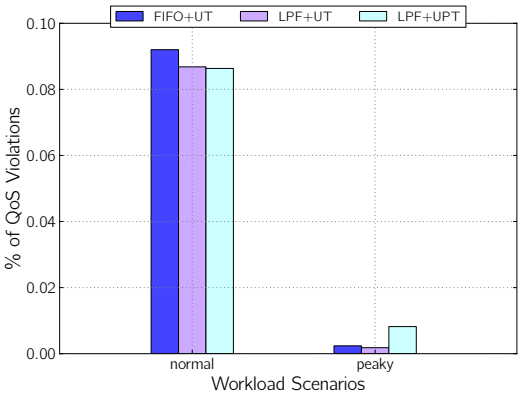


Figure 10: **Unbounded:** Percentage of QoS violations for FIFO+UT, LPF+UT, LPF+UPT.

As expected, the number of QoS violations is considerably smaller in these scenarios. Figure 10 shows that the occurrence of such events is extremely rare in both workloads ($< 1\%$). One can also observe a potentially statistically negligible, but nevertheless interesting, phenomenon in this figure: all the strategies yielded more QoS violations in the Normal workload than in Peaky. Moreover, differences on the percentage of dissatisfaction presented in Figure 12 reinforces this apparent contradiction. The explanation for these results lay in an observation that was irrelevant in scenarios where QoS violations take place frequently: although its changes are more abrupt, Peaky is smoother, in the sense that

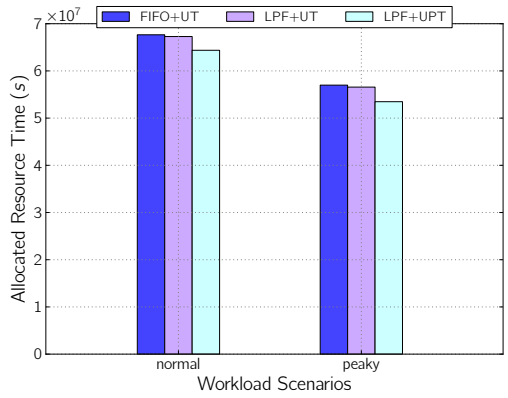


Figure 11: **Unbounded:** Comparison of resource-time usage for FIFO+UT, LPF+UT, LPF+UPT.

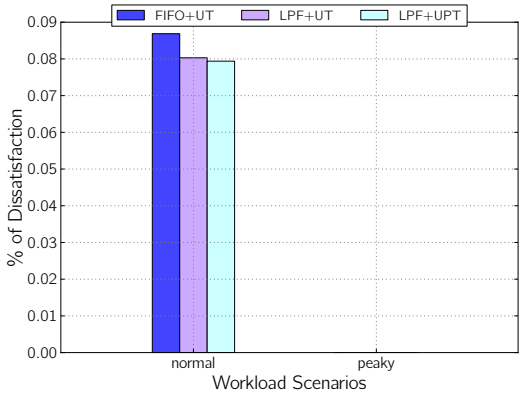


Figure 12: **Unbounded:** Percentage of user dissatisfaction for FIFO+UT, LPF+UT, LPF+UPT.

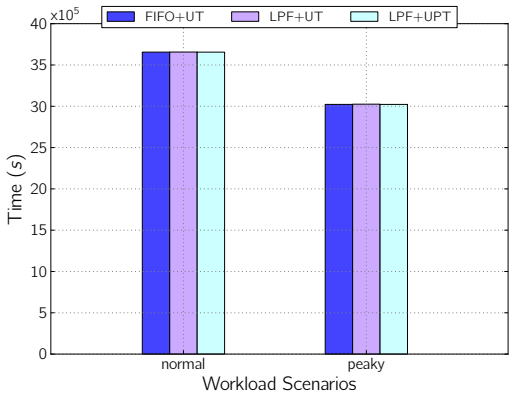


Figure 13: **Unbounded:** Aggregate request slowdown for FIFO+UT, LPF+UT, LPF+UPT.

modifications on its upwards/downward orientation are less frequent than in Normal (6 against 15); QoS violations are more likely to occur during these shifts, which justifies the observed difference. Additionally, Figure 10 shows that FIFO+UT delivered less QoS violations than the other two strategies for the Peaky workload; this happened because patience-aware strategies focus on the minimization of user dissatisfaction rather than QoS violations, and sacrificing the latter may be necessary in order to excel on the first.

Conversely, results presented in Figure 12 suggest that the impact of QoS violations on end-user patience on Normal was considerably larger than on Peaky. Direct comparison with Figure 10 suggests that the number of violations play a major role on this phenomenon, so we investigated the aggregate request slowdown.

The results are presented in Figure 13 and show that the values for Peaky and Normal are close; consequently, individual QoS violations typically involved much larger delays on Peaky than on Normal. From this, we conclude that having several small delays is more harmful to user dissatisfaction than a few long delays. Finally, Figure 11 reinforces the benefit of using a patience-aware auto-scaling triggering strategy. Whereas user dissatisfaction levels for all configurations are basically the same in both workloads, LPF+UPT consumed 9% less resources than the other strategies.

To evaluate the impact of provisioning time on the auto-scaling strategies described in this work, we performed an experiment with provisioning time of $10 \pm 5$ seconds and 10,000 as a maximum number of end-users — that is, $|\mathcal{U}| = 10,000$ when computing the number of users in the system $a_t$ at time $t$. The minimum number of resources, $b$, was set to 450. This scenario presents demand peaks that exceed 2,000 VMs. Figure 14 and Figure 15 summarise the results for allocated-resource time and percentage of dissatisfactions, respectively.

We can observe that the patterns identified in previous experiments hold under this last scenario; there is a reduction in the amount of allocated resource time resulting into a small increase in the number of requests that suffer from insufficient QoS. Further investigation revealed that this is the case because, although resources can be provisioned quickly, LPF delays auto-scaling operations by exploiting users patience levels, eventually leading to dissatisfactions as resource utilisation and triggering of auto-scaling decisions are taken at discrete intervals. Although the patience-based auto-scaling strategy could be made more responsive to variations in resource utilisation, that would lead to an increase in the cost with infrastructure as resources would be activated and released more often, and eventually render the consideration of users' patience levels in auto-scaling unusable.

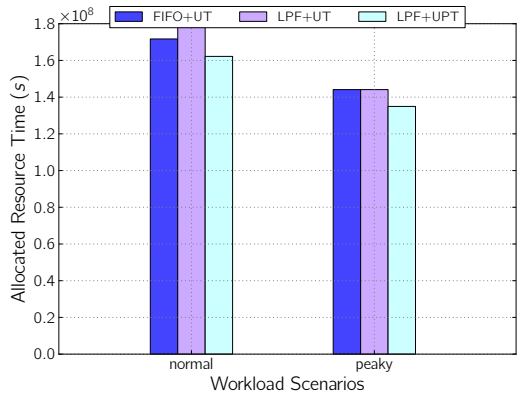


Figure 14: **Unbounded:** Comparison of resource-time usage for FIFO+UT, LPF+UT, LPF+UPT under provisioning time of 10±5 seconds and $|\mathcal{U}| = 10,000$ users.

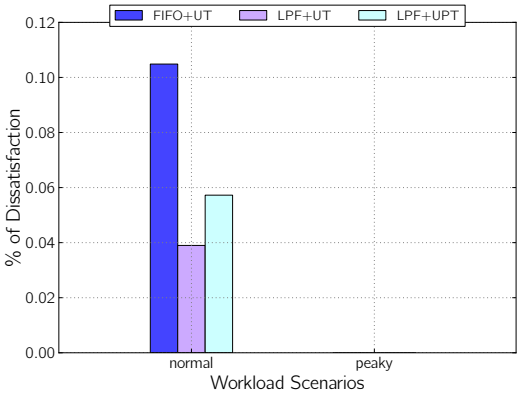


Figure 15: **Unbounded:** Percentage of user dissatisfaction for FIFO+UT, LPF+UT, LPF+UPT under provisioning time of 10±5 seconds and $|\mathcal{U}| = 10,000$ users.

## 5. Related Work

Projects related to our work fall into categories of scheduling, user behaviour, and cloud computing auto-scaling. Scheduling is a well-studied topic in several domains for which a large number of theoretical problems, solutions, heuristics, and practical applications have been considered [13, 14]. Some of the used algorithms include FIFO, priority-based, deadline-driven, hybrid approaches that use backfilling techniques [15], among others [16, 17]. In addition to priorities and deadlines, other factors have been considered, such as fairness [18], energy-consumption [19], and context-awareness [9]. Moreover, utility functions were used to model how the importance of results to users varies over time [20, 21].

User behaviour has been explored for optimising resource management in the context of Web caching and page pre-fetching [22, 23, 24, 25, 26, 27]. The goal in previous work was to understand how users access Web pages, investigate how tolerant users are to delays, and pre-fetch or modify page content to enhance user experience. Techniques in this area focus mostly on adapting Web content and minimising response time of user requests. Service research has also investigated the impact of delays in users' behaviour. For instance, Taylor described the concept of delays and surveyed passengers affected by delayed flights in order to test various hypotheses [28]. Brown *et al.* and Gans *et al.* investigated the impact of service delays in call centres [29, 30]. In behavioural economics, Kahneman and Tversky [10] introduced Prospect Theory to model how people make choices in situations that involve risk or uncertainty. Netto *et al.* introduced a scheduling strategy that considers information on how quickly users consume results generated by service providers [7]. Our previous work investigated the scheduling of user requests considering their patience and expectations, but with no auto-scaling of cloud resources [6].

Shen *et al.* introduced a system for automating elastic resource scaling for cloud computing [31]. Their system does not require prior knowledge on the applications running in the cloud. Other projects consider auto-scaling in different scenarios and proposed several approaches, such as modifying the number of resources allocated for running MapReduce applications [32, 33], comparing vertical versus horizontal auto-scaling [34], minimising operational costs [35], and providing integer model based auto-scaling [36] .

Unlike previous work, our proposed auto-scaling technique considers information on user patience while interacting with a service offered by a service provider using resources from a cloud infrastructure.

## 6. Conclusions and Future Work

In this article, we extended our investigation on patience-based algorithms for auto-scaling strategies. Traditional mechanisms are based solely on resource utilisation information and other system metrics, ignoring actual users' needs with respect to their desired QoS levels. As a consequence, resources are over-provisioned under scenarios where they are not strictly needed; that is, where additional resources do not lead to significant improvement in QoS.

We conducted extensive experiments using FIFO and Lowest Patience First as scheduling algorithms, and auto-scaling triggers that rely either on utilisation or

user patience and utilisation. Our experiments considered scenarios where the maximum number of resources could be either bounded or unbounded, and different versions of the auto-scaling algorithms were designed to cope with both possibilities. Our experiments considered non-trivial (*i.e.*, larger than 0) provisioning time.

Experimental results suggest that patience-based strategies can indeed provide significant economic gains to service providers. Namely, we observed that these strategies were able to reduce resource-time usage by approximately 9% while keeping QoS at the same level as those based solely on utilisation. We conclude that the identification of unnecessary buffers in QoS levels that will not necessarily improve user experience in cloud environments is a key element for service providers, as it may enable savings in infrastructure costs and bring competitive advantage for adopters.

We believe the proposed strategies fill an important gap in the literature. As sensors and Internet of Things become more pervasive, more data will be available that can be used to determine factors that influence users' patience. By using this data, one could certainly specialise and improve the techniques presented here. With respect to resource management, the strategies presented in this work handled horizontal resource elasticity and assumed that resources are pairwise indistinguishable. In future work, we aim to relax such assumptions, handling vertical elasticity and the allocation of multiple types of virtual machine instances.

## Acknowledgements

Some experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the Inria ALADDIN development action with support from CNRS, RENATER and several Universities, as well as other funding bodies (see https://www.grid5000.fr). This work has also been supported and partially funded by FINEP / MCTI, under subcontract no. 03.14.0062.00.

## References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia, Above the clouds: A Berkeley view of Cloud computing, Technical report UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, USA (February 2009).

[2] R. Barga, D. Gannon, D. Reed, The client and the cloud: Democratizing research computing, IEEE Internet Computing 15 (1) (2011) 72–75.

[3] J. Ramsay, A. Barbesi, J. Preece, A psychological investigation of long retrieval times on the world wide web, Interacting with computers 10 (1) (1998) 77–86.

[4] The communications market report: United kingdom (August 2013).

[5] F. Venturini, Five insights into consumers' online video viewing and buying habits, Outlook, Point of View (July 2013).

[6] C. Cardonha, M. D. Assunção, M. A. S. Netto, R. L. F. Cunha, C. Queiroz, Patience-aware scheduling for cloud services: Freeing users from the chains of boredom, in: 11th International Conference on Service Oriented Computing (ICSOC'13), 2013.

[7] M. A. S. Netto, et al., Leveraging attention scarcity to improve the overall user experience of cloud services, in: Int. Conf. on Network and Service Management (CNSM'13), 2013.

[8] R. L. F. Cunha, M. D. de Assuncao, C. Cardonha, M. A. S. Netto, Exploiting user patience for scaling resource capacity in cloud services, in: 7th IEEE International Conference on Cloud Computing (IEEE Cloud 2014), IEEE, 2014, pp. 448–455.

[9] M. D. de Assuncao, M. A. S. Netto, F. Koch, S. Bianchi, Context-aware job scheduling for cloud computing environments, in: IEEE/ACM Fifth International Conference on Utility and Cloud Computing (UCC 2012) Workshops, IEEE Computer Society, Washington, USA, 2012, pp. 255–262. doi:10.1109/UCC.2012.33.
URL `http://dx.doi.org/10.1109/UCC.2012.33`

[10] D. Kahneman, A. Tversky, Prospect theory: An analysis of decision under risk, Econometrica: Journal of the Econometric Society (1979) 263–291.

[11] M. A. S. Netto, C. Cardonha, R. L. F. Cunha, M. D. de Assuncao, Evaluating auto-scaling strategies for cloud computing environments, in: 22nd IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2014), IEEE, 2014.

[12] M. Mao, M. Humphrey, A performance study on the VM startup time in the cloud, in: IEEE 5th International Conference on Cloud Computing (CLOUD 2012), 2012, pp. 423–430.

[13] J. Blazewicz, K. Ecke, G. Schmidt, J. Weglarz, Scheduling in Computer and Manufacturing Systems, 2nd Edition, 1994.

[14] M. L. Pinedo, Planning and Scheduling in Manufacturing and Services, 2nd Edition, 2007.

[15] D. Tsafrir, Y. Etsion, D. G. Feitelson, Backfilling using system-generated predictions rather than user runtime estimates, IEEE Transactions on Parallel and Distributed Systems 18 (6) (2007) 789–803.

[16] T. D. Braun, et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed computing.

[17] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, P. Wong, Theory and practice in parallel job scheduling, in: Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'97), 1997.

[18] N. D. Doulamis, A. D. Doulamis, E. A. Varvarigos, T. A. Varvarigou, Fair scheduling algorithms in grids, IEEE Transactions on Parallel and Distributed Systems 18 (11) (2007) 1630–1648.

[19] J.-F. Pineau, Y. Robert, F. Vivien, Energy-aware scheduling of bag-of-tasks applications on master–worker platforms, Concurrency and Computation: Practice and Experience 23 (2) (2011) 145–157.

[20] C. B. Lee, A. Snavely, Precise and realistic utility functions for user-centric performance analysis of schedulers, in: Int. Symp. on High-Performance Distributed Computing (HPDC'07), 2007.

[21] A. AuYoung, et al., Service contracts and aggregate utility functions, in: 15th IEEE Int. Symp. on High Performance Distributed Computing (HPDC'06), 2006.

[22] D. F. Galletta, R. M. Henry, S. McCoy, P. Polak, Web site delays: How tolerant are users?, Journal of the Association for Information Systems 5 (1) (2004) 1–28.

[23] F. Alt, A. Sahami Shirazi, A. Schmidt, R. Atterer, Bridging waiting times on web pages, in: 14th Int. Conf. on Human-computer interaction with mobile devices and services (MobileHCI'12), ACM, New York, NY, USA, 2012, pp. 305–308.

[24] R. Atterer, M. Wnuk, A. Schmidt, Knowing the user's every move: user activity tracking for website usability evaluation and implicit interaction, in: 15th Int. Conf. on World Wide Web (WWW'06), ACM, New York, NY, USA, 2006, pp. 203–212.

[25] C. R. Cunha, C. F. B. Jaccoud, Determining www user's next access and its application to prefetching, in: 2nd IEEE Symp. on Computers and Communications (ISCC '97), Washington, DC, USA, 1997, pp. 6–.

[26] N. Bhatti, A. Bouch, A. Kuchinsky, Integrating user-perceived quality into web server design, Computer Networks 33 (1) (2000) 1–16.

[27] A. Bouch, A. Kuchinsky, N. Bhatti, Quality is in the eye of the beholder: meeting users' requirements for internet quality of service, in: SIGCHI conference on Human factors in computing systems, ACM, 2000, pp. 297–304.

[28] S. Taylor, Waiting for service: the relationship between delays and evaluations of service, The Journal of Marketing (1994) 56–69.

[29] L. Brown, N. Gans, A. Mandelbaum, A. Sakov, H. Shen, S. Zeltyn, L. Zhao, Statistical analysis of a telephone call center: A queueing-science perspective, Journal of the American Statistical Association 100 (2005) 36–50.

[30] N. Gans, G. Koole, A. Mandelbaum, Telephone call centers: Tutorial, review, and research prospects, Manufacturing & Service Operations Management 5 (2) (2003) 79–141.

[31] Z. Shen, S. Subbiah, X. Gu, J. Wilkes, Cloudscale: elastic resource scaling for multi-tenant cloud systems, in: 2nd ACM Symposium on Cloud Computing, ACM, 2011, p. 5.

[32] Y. Chen, S. Alspaugh, R. Katz, Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads, Proc. VLDB Endow.

[33] Z. Fadika, M. Govindaraju, DELMA: Dynamically ELastic MapReduce Framework for CPU-Intensive Applications, in: 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'11), 2011, pp. 454–463.

[34] M. Sedaghat, F. Hernandez-Rodriguez, E. Elmroth, A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling, in: ACM Cloud and Autonomic Computing Conference (CAC'13), ACM, 2013.

[35] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC), IEEE, 2011, pp. 1–12.

[36] M. Mao, J. Li, M. Humphrey, Cloud auto-scaling with deadline and budget constraints, in: 11th IEEE/ACM International Conference on Grid Computing (GRID), IEEE/ACM, 2010, pp. 41–48.