Queueing Systems S.I. in Cloud Computing manuscript No.
(will be inserted by the editor)

Admit or Preserve? Addressing Server Failures in Cloud Computing Task Management

Nadav Lavi · Hanoch Levy

Received: date / Accepted: date

Abstract Cloud computing task management has a critical role in the efficient operation of the cloud resources, i.e., the servers. The task management handles critical and complicated decisions, overcoming the inherent dynamic nature of cloud computing systems and the additional complexity due to the large magnitude of resources in such systems (tens of thousands of servers). Due to the fact that servers may fail, task management is required to conduct both task admissions and task preservations decisions. Moreover, both these decisions require considering future system trajectories and the interplay between preservation and admission.

In this paper we study the combined problem of task admission and preservation in a dynamic environment of cloud computing systems through analysis of a queueing system based on a Markov Decision Process (MDP). We show that the optimal operational policy is of a double switching-curve type. On its face value, the extraction of the optimal policy is rather complicated, yet our analysis reveals that the optimal policy can be reduced to a single rule, since the rules can be effectively decoupled. Based on this result, we propose two heuristic approaches that approximate the optimal rule for the most relevant system settings in cloud computing systems. Our results provide a simple policy scheme for the combined admission and preservation problem that can be

Nadav Lavi
School of Electrical Engineering, Tel-Aviv University
General Motors Advanced Technical Center - Israel, Hamada 7, Herziliya 46733, P.O.B. 12091, Israel
Tel.: +972-9-9720628
Fax: +972-9-9720601
E-mail: nadavlavi@post.tau.ac.il, nadav.lavi@gm.com

Hanoch Levy
School of Computer Science, Tel-Aviv University, P.O.B. 39040, Ramat Aviv, Tel Aviv 69978, Israel
Tel.: +972-3-640-7401
Fax: +972-3-640-9357
E-mail: hanoch@cs.tau.ac.il

applied in a complex cloud computing environments, and eliminate the need for sophisticated real-time control mechanisms.

Keywords Markov Decision Processes · Admission Control · Task Management · Task Preservation · Cloud Computing

Mathematics Subject Classification (2000) 90C40 · 90B22 · 93E03

1 Introduction

Cloud computing reliability is an acute problem for the Cloud Computing Centers (CCC) operator. Although cloud computing is perceived as a reliable always on service, behind the scenes tens of thousands of general computing platforms, i.e., servers, operate under continuous load due to endless service demand. These servers may fail [6, 7, 10, 35] due to either a component failure, such as hard-drive or memory, or due to a software glitch, for example in the virtualization layer. With the large number of servers and components in CCC, failure probability becomes high and with a significant impact on system performance. Such failures derive additional complexity on the CCC task management mechanism, as it is now being required to handle both admission of new tasks and preservation of existing tasks that their servers failed. Similar task management challenges are faced by service providers who provide services to customers by renting resources from the CCC.

The questions of admission and preservation seem to be strongly coupled. Should a new task be admitted and be served by an available idle server, or should that server be held for preserving existing tasks in the event their server fails? Similarly, should a task be preserved upon the failure of its server, thus requiring the allocation of an idle server, or the idle server be held for new arrivals? The admission and preservation decisions of the task management are not trivial, as they need to consider future projections of the system behavior. Hence, not only it requires the considerations of the system settings, but also the interplay between admission and preservation. As a result, an admission decision of a new task needs to account for future failures. Thus, the CCC task management faces a tradeoff between admitting new tasks (which will be beneficial in the immediate future) versus keeping certain number of servers idle for future failures (which will be beneficial in the long run).

In this paper we construct a new modeling framework which provides a holistic optimization scheme for the combined problems of new task admission and existing task preservation. We base our model on a queuing-loss multi-server system with failures and address the optimal operation of the CCC. We formulate a reward and cost model that incorporates rewards per task admission, as well as costs due to server assignment (upon admission or preservation), task rejection (when admission is not feasible) and task drop (when it is impossible to preserve a task). We further show how this model can be reduced to a cost minimization problem. Based on this new formulation, we

evaluate the optimal policy of the combined admission/rejection and preservation/dropping decisions using a Markov Decision Process (MDP) [29]. For the problem composition as an MDP, we construct a new operator: the *preservation operator*. We investigate the various properties of the new operator for both our specific analysis and for other system settings.

On its face value, the problem seems to be difficult as it involves both a three dimensional space (number of busy servers, idle servers and failed servers) and mutually affecting decisions (admission and preservation). Yet our analysis reveals that the problem simplifies as the decision rules can be decoupled. Our analysis is based on two main stages: in the first stage we show that the optimal admission and preservation decision rules can be characterized as a double *switching-curve*, one for admission and one for preservation. That is, the optimal decision is to admit (preserve) a new (existing) task if the number of failed servers is below a certain threshold which depends on the number of active servers in the system. The second stage of the analysis reveals that unexpectedly the two rules can be effectively be *decoupled*. That is, we show that the problem can be reduced to a set of problems, in each of which one rule is of a switching-curve type and the other rule takes the trivial form.

The outcome of our analysis enables us to derive a simple policy based on the system cost-structure, which simplifies the CCC management. This is due to the fact that task management is now based only on a single rule. Hence, our scheme significantly simplifies the task management operation as it eliminates the need for complex real-time mechanisms. Yet, as the extraction of the switching-curve requires numerical computations, which strongly depends on the state-space, we further propose two heuristic approximation approaches to derive sub-optimal, yet effective, policies. We numerically study the performance of the system and the characteristics of the optimal operational rule in various system settings and cost structures. We demonstrate that the proposed heuristic rules are quite accurate for a very wide and practical set of cases. Thus, providing practical tools to extract policies for cloud task management.

This paper is organized as follows: Section 2 describes previous related work. In Section 3 we describe the tradeoffs between the admission and preservation decision in CCC. In Section 4 we model the system. In Section 5 we formulate the problem as an MDP optimization problem. In Section 6 we investigate the properties of the newly composed preservation operator. In Section 7 establish the fundamental properties of the value function and show that the optimal admission and preservation schemes are characterized by a double switching curve. In Section 8 we analyze the joint admission and preservation rules and show that effectively they can be reduced to a collection of problems each with a single switching curve. Section 9 presents simulation results and investigation of how various parameters impact the admission and preservation decisions. We then propose, in Section 10, two heuristic approximation approaches for large state space systems and investigate their performance. In Section 11 we describe additional use-cases for our scheme and future research directions, and conclude.

2 Previous Work

Cloud computing infrastructure reliability was investigated in both hardware (server and its components) and software/Virtual-Machines (VMs) aspects. [6, 10, 35] investigated statistical data from operational cloud systems. In [7] statistical data, based on cloud deployment experience, is detailed including hardware failure rate per potential reason and computing resources downtime duration. VM failures impact was investigated in [24], where a computing model is composed and used for comparison between VM replication algorithms which target to increase the cloud failure resiliency. Other papers that investigated resiliency in cloud computing and suggested techniques to overcome them are [12, 17, 19, 20]. We can clearly derive that although cloud computing is perceived as reliable always-on infrastructure, there is a high probability of component failures which requires to properly manage and handle tasks and overcome such incidents.

Another potential solution for reliability and resiliency is based on redundant allocations, i.e., allocating several resources, either physical (servers) or virtual (VMs), to the same task. Both [3, 7, 18] investigated the k-resiliency approach to overcome failures, more specifically k failures, through the use of redundant VM allocations. We note that the k-resiliency techniques address solely the admission decision. Preservation decision are not relevant in such schemes as the system pre-allocates sufficient, and potentially more than actually required, servers once the task is admitted to the system. Hence, these techniques are potentially inefficient in the utilization of the CCC resources.

Admission control (or resource control) for cloud computing centers was addressed in the past by [9,34]. Both papers consider heterogeneous tasks with different characteristics and optimize operational aspects such as minimizing the rejection probability or maximizing the revenues. In addition, both papers do not include a failure model and do not address task preservations. Another cloud related domain in which admission control is important is Hybrid Clouds [13, 33]. In Hybrid Clouds the infrastructure is composed of both private and public cloud computing centers. Diverting (offloading) a task from the private to the public cloud implies additional cost to the private cloud operator. We note that both [9] and [33] use MDP in their modeling of dynamic cloud systems, and techniques from the latter are employed in this paper.

Admission-control was also investigated through the use of tools from queueing systems in several similar settings, i.e., admission-control in queueing-loss systems and admission-control with failures. Admission-control in queueing-loss systems was first addressed in [22] which studied the case of optimal admission with n identical servers and m types of customer classes defer only by their reward. Later [16] generalized the problem. [1, 15] also studied the admission-control problem with multiple types of customers.

Systems with server failures were first addressed in [23, 25] focusing on homogeneous servers. [8] investigated the optimal customer routing in a two heterogenous servers system where the servers differ in their service rate and

reliability attributes. Later [27, 28] provided a more complete investigation of the same problem.

We conclude and emphasize that this work is the first to deal with the combined problem of admission and preservation and its optimal dynamic control; prior works dealt either with admission control or with server failures but not with the combined problem. The presence of server failures in a queueing-loss system introduces a unique settings in which the admission decision affects not only the current system status but also future admission and more importantly preservation decisions. While admission policies related to the papers mentioned above consider only the impact of admission versus rejection, the admission policy required for the combined problem incorporates the system dynamics and the potential impact of server failures on the operational cost. Employing admission policies without considering server failures and tasks preservation will eventually result in high penalties due to lack of available resources to handle tasks preservation.

3 Task Preservation and Task Admission: An Inherently Combined Problem

We will illustrate the problem faced by the Cloud Computing task management throughout the lifetime of a task in the system. We assume that the system gains a reward upon a new task admission, and incurs a penalty if the task cannot be admitted, i.e., rejected, or if it cannot be preserved upon server failure, i.e., dropped. Furthermore, we assume that server allocation (upon admission and preservation) derives operational costs, for example due to the environment setup. Upon server failure, the task management mechanism faces a decision problem of whether to drop the task and accept the penalty, or rather preserve the task and incur additional operational cost. Yet, if the task is preserved, there is a potential that the system will be mandated to drop the task in the future in case of sequential failures. Hence, incurring both the operational expenses and the drop penalty. Another decision faced by the task management, considers the treatment of new task arrivals; The task management then needs to consider whether it is beneficial to accept the task and therefore allocate an available computing resource at present time, or save this resource for the future in order to overcome potential active-server failures. Using the resource may benefit an instantaneous reward, but might eventually cause a more costly penalty in the case of the need to drop an existing task due to server failure and lack of available resources.

The complexity in answering these questions increases when we consider the potential difference of future rejection, task drop and server allocation costs. In this paper we will address these questions, and provide a holistic framework for the combined admission and preservation problem.

4 System Model

We model a CCC as a queueing-loss system with M servers. The assumption of a fixed number of servers is reasonable as CCC is planned in advance for a certain capacity. The selection of a queueing-loss system is based on the fact that an arriving task will usually expect an immediate response. The fact that a task may be rejected means that it will be transferred to another computing facility. Tasks arrive to the CCC according to a Poisson process with rate λ. A task is either admitted and allocated a server or rejected. For simplicity we assume a single task type, i.e., single priority level, and all servers are identical (in terms of resources). If a task is admitted then the system gains a reward $R > 0$ and incurs an operational cost of $C_a \geq 0$ due to the server allocation. If the task is rejected then the system incurs a rejection penalty of $C_p \geq 0$. An admitted task service time follows exponential distribution with parameter μ.

Servers in the system are in one of three states: active, idle, and failed. An active server is a server which is serving a task, an idle server is an available server ready to serve a task when needed, and a failed server is a server waiting for repair and cannot be used before that. We note that both active and idle servers may fail. The duration after which an active or idle server may experience failure is exponentially distributed with parameter μ_f. If an active server fails, the task can be maintained in the system and preserved using a new server with an additional server allocation cost $C_a \geq 0$, or it can be dropped, i.e., stopped being served, with penalty of $C_d \geq 0$. Similarly, to the admission scenario, we assume that if a task cannot be preserved it is transferred to another computing facility. Server replacements are handled one after the other with exponentially distributed duration with parameter μ_r. The behavior of the system is illustrated in Figure 1.

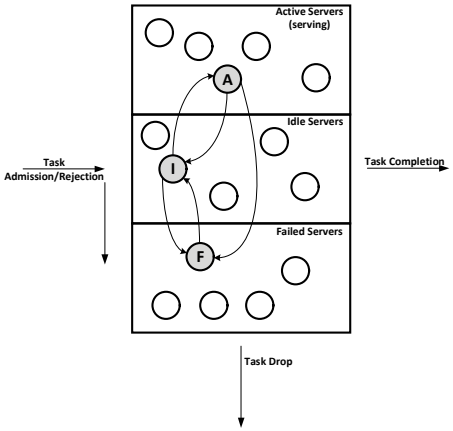


Fig. 1: System behavior - tasks and servers.

The goal of the CCC task management, i.e., the combined admission and preservation decision rules, is to maximize the system *revenue rate*, which equals to the *reward rate* minus the operational costs (tasks' admission, rejection, preservation and drop).As the system *reward rate* is fixed and defined by λR, similarly to [26], we can formulate an equivalent cost minimization problem and examine the expected total discounted cost over an infinite horizon. We note that in this case, the rejection cost, denoted as C_r, is consisted of the rejection penalty (C_p) plus R (returning the reward).

Although we do not bound our analysis to specific conditions between the various cost parameters, it is reasonable to assume that for cloud computing an adequate cost structure is the one in which $C_r > R$ and $C_d > C_r$. Meaning, upon rejection the system not only loose the reward R but also incurs an additional penalty. Furthermore, dropping an existing task has more impact as it incurs a larger penalty compared to rejecting a new task. We note that we also address other cost structures, such as $C_r > C_d$ in which the unique settings derive different behavior than the former cost structure.

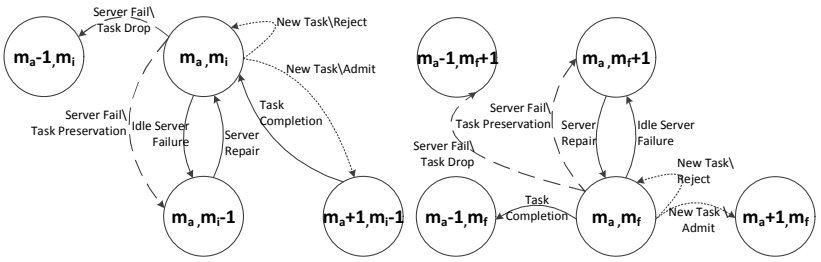
We note that even though our model considers a single task allocation per server, the proposed scheme can be easily extended to multi-task support (by a single server) through the use of task grouping. In cloud computing multi-task is supported by Virtual Machines (VMs) running on the same server. Hence, if a server fails all VMs die and therefore all the tasks serviced by this server require preservation.

5 MDP Formulation

As indicated above the total number of servers in the system is M. We denote the number of active, idle and failed servers as m_a, m_i, and m_f, respectively. Therefore:

$$m_a + m_i + m_f = M,\ m_a, m_i, m_f \geq 0 \tag{1}$$

We note that due to (1), it is sufficient to use two parameters (out of the three) to describe the system state space in the MDP formulation. Although the state space formulation using (m_a, m_i) $(m_a + m_i \leq M)$, i.e., using the active and idle servers, is an intuitive approach, we choose to represent the system state space using the active and failed servers (m_a, m_f) $(m_a + m_f \leq M)$. The rational in this transformed representation is our goal to prove certain structural properties of the optimal solution. Therefore, we strive to leverage specific characteristics of the value function, namely nondecreasing, convexity and supermodularity. Using idle servers in the representation, the value function is obviously decreasing due to the fact that as m_i increases there are more available servers to admit and preserve tasks, and therefore the system cost per admission or preservation event decreases. The behavior of the state space with the representation using active and idle servers is illustrated in Figure 2a. The transformed state space, i.e., active and failed servers, is illustrated in Figure 2b.



(a) Original: # of active and idle. (b) Transformed: # of active and failed.

Fig. 2: State transitions: original and transformed.

As investigation of finite state space MDP is complex due to the boundary effects [15], we formulate our state space as an infinite state space MDP, and apply methods from [14] and [1] to cover all boundary conditions, i.e., preventing the system to continue beyond $m_a + m_f = M$.

We note that the events corresponding to task completion and server failure depend on the system state and are not i.i.d. To overcome this we apply the well-known *uniformization* method [14] and inject *virtual* server failure and task completion events. Due to the change to an infinite state-space, we bound the overall task completion events by $M\mu$, and the overall server failures by $2M\mu_f$ (M failure events per dimension). As a result we can say that the system events (both real and virtual) follow a Poisson process with rate $\Lambda = \lambda + \mu_r + M\mu + 2M\mu_f$.

We can now formulate the problem as a *discrete time* MDP [29]:

$$
\begin{aligned}
V(m_a, m_f) &= \delta\frac{\lambda}{\Lambda}\min\{C_a + V(m_a+1, m_f), C_r + V(m_a, m_f)\} \\
&+ \delta\frac{\min\{m_a, M\}\mu}{\Lambda}V((m_a-1)^+, m_f) \\
&+ \delta\frac{\mu_r}{\Lambda}V(m_a, (m_f-1)^+) \\
&+ \delta\frac{(M-m_a-m_f)^+\mu_f}{\Lambda}V(m_a, m_f+1) \\
&+ \delta\frac{\min\{m_a, M\}\mu_f}{\Lambda}\min\{C_a + V(m_a, m_f+1), C_d + V((m_a-1)^+, m_f+1)\} \\
&+ \delta\frac{(M-\min\{m_a, M\} + \min\{M, m_a+m_f\})\mu_f}{\Lambda}V(m_a, m_f) \\
&+ \delta\frac{(M-m_a)^+\mu}{\Lambda}V(m_a, m_f) \\
&+ K(m_a + m_f - M)^+,\ m_a \geq 0, m_f \geq 0
\end{aligned} \tag{2}
$$

where δ is the discount factor ($0 < \delta \leq 1$), Λ (the combined events rate) as defined above, and K is a sufficiently large constant such that it prevents entering states that are beyond system boundaries. We emphasize that the

inclusion of K in V does not change the results within the original finite state space. It is important to note that due to the fact that the original state space and action space in each state are finite, our results, i.e., the admission and preservation policies, also hold for the long-run average cost ($\delta = 1$).

We note that V uniquely solves the Bellman optimality equation, and through V optimal admission and preservation policies can be derived.

We now apply a method presented in [14] and construct our value function using a set of operators. Through the use of operators and their characteristics we will be able later to prove certain properties of the value function, which will then assist in understanding the optimal admission and preservation policies. We define the following operators for $f : \mathbb{N}_0^2 \to \mathbb{R}$:

$$T_{CA(1)}f(x_1, x_2) = \min\{c + f(x_1, x_2), c' + f(x_1 + 1, x_2)\} \tag{3}$$

$$T_{D1(2)}f(x_1, x_2) = f(x_1, (x_2 - 1)^+) \tag{4}$$

$$T_{D(1,N)}f(x_1, x_2) = \min\{x_1, N\}\mu f(x_1 - 1, x_2) + (N - x_1)^+\mu f(x_1, x_2) \tag{5}$$

$$T_K f(x_1, x_2) = K(x_1 + x_2 - N)^+ \tag{6}$$

$$T_{A(2)}f(x_1, x_2) = f(x_1, x_2 + 1) \tag{7}$$

where, $c, c' \in \mathbb{R}$.

We note that the aforementioned operators are well known and previously investigated in the literature [14] (and references therein). Yet, our scheme introduces a new operator that handles the decision upon active server failures, the preservation operator. We define this operator as follows:

$$T_{CP}f(x_1, x_2) = \min\{c + f(x_1, x_2 + 1), c' + f(x_1 - 1, x_2 + 1)\} \tag{8}$$

We now rewrite (2) using the aforementioned operators, and define in (9) a new operator: T.

$$\begin{aligned} TV(x_1, x_2) = {} & \delta\frac{\lambda}{\Lambda}T_{CA(1)}V(x_1, x_2) \\ & + \delta\frac{\min\{m_a, M\}\mu}{\Lambda}T_{D(1,M)}V(x_1, x_2) \\ & + \delta\frac{\mu_r}{\Lambda}T_{D1(2)}V(x_1, x_2) \\ & + \delta\frac{(M - x_1 - x_2)^+\mu_f}{\Lambda}T_{A(2)}V(x_1, x_2) \\ & + \delta\frac{\min\{x_1, M\}\mu_f}{\Lambda}T_{CP}V(x_1, x_2) \\ & + \delta\frac{(M - \min\{M, x_1\} + \min\{M, x_1 + x_2\})\mu_f}{\Lambda}V(x_1, x_2) \\ & + \delta\frac{(M - x_1)^+\mu}{\Lambda}V(x_1, x_2) \\ & + T_K V(x_1, x_2) \end{aligned} \tag{9}$$

Then the Bellman equation reads $TV = V$.

We note that in our problem formulation $c = C_a$ in both $T_{CA(1)}$ and T_{CP}, and $c' = C_r$ and $c' = C_d$ in $T_{CA(1)}$ and T_{CP}, respectively.

In the next section we prove certain properties of the new preservation operator, and later we show how these properties, combined with the properties of the other operators, enable us to prove the optimality of the admission and preservation policies.

6 Properties of the Preservation Operator

In this section we establish the fundamental properties of the preservation operator. Later we will use elements from the properties to derive the optimal operational rule.

Throughout this paper we follow the property definitions from [14]. Denoting f as a function from $\mathbb{N}_0^m$ to $\mathbb{R}$ and $X \in \mathbb{N}_0^m$ ($m \in \mathbb{N}$), we now define the following properties:

- $f(X)$ is nondecreasing in x_i if $f(X) \leq f(X + e_i)$
- $f(X)$ is upstream-increasing in x_i if $f(X + e_{i+1}) \leq f(X + e_i)$, $1 \leq i < m$
- $f(X)$ is convex in x_i if $2f(X + e_i) \leq f(X) + f(X + 2e_i)$
- $f(X)$ is supermodular in x_i, x_j $(1 \leq i < j \leq m)$ if $f(X + e_i) + f(X + e_j) \leq f(X + e_i + e_j) + f(X)$
- $f(X)$ is submodular in x_i, x_j $(1 \leq i < j \leq m)$ if $f(X) + f(X + e_i + e_j) \leq f(X + e_i) + f(X + e_j)$
- $f(X)$ is super-convex in x_i, x_j $(1 \leq i, j \leq m,\ i \neq j)$ if $f(X + e_i + e_j) + f(X + e_i) \leq f(X + 2e_i) + f(X + e_j)$
- $f(X)$ is sub-convex in x_i, x_j $(1 \leq i, j \leq m,\ i \neq j)$ if $f(X + e_i + e_j) + f(X + e_i) \leq f(X + 2e_i + e_j) + f(X)$

where e_i is a vector of same dimension as X with all zeros and a 1 at the i'th location.

Similarly to [14] we denote nondecreasing, upstream-increasing, and convex in x_i as **I(i)**, **UI(i)**, and **Cx(i)**, respectively. If the properties exist in all dimensions we can simply use the notation **I**, **UI**, and **Cx**. In addition, we denote supermodular, submodular, super-convex, and sub-convex in x_i and x_j as **Super(i,j)**, **Sub(i,j)**, **SuperC(i,j)**, and **SubC(i,j)**, respectively. If the properties exist in all of the potential dimension combination (under the conditions stated above per property) we use the notation **Super**, **Sub**, **SuperC**, and **SubC**.

We now prove the aforementioned properties for the preservation operator T_{CP}. We keep the definitions from [14] where, $T : A \to B$ means that if $f \in A$ then $Tf \in B$ ($A \subset B$). If T maintains all properties of f we say that T preserves f. We focus on the two dimensional case (x_1, x_2) as it is the state-space of our CCC.

As indicated above, for the purpose of our specific analysis we are interested in a subset of the aforementioned properties. However, in order to

provide a more comprehensive analysis of the new preservation operator and its potential in the investigation of other systems, we prove entire set of properties in the following sections. We divide the properties into two groups: the properties essential to our value function investigation (non-decreasing, convex, and supermodular), and the additional properties maintained by the new preservation operator and can be used in other types of systems.

6.1 Essential Properties Maintained by the Preservation Operator

Theorem 1 $T_{CP}f(x_1, x_2) : \boldsymbol{I} \longrightarrow \boldsymbol{I}$ *(non-decreasing)*

Proof We assume that $f \in \mathbf{I}$. We need to show that for any of the decisions taken by the preservation operator (on the RHS of the definition), i.e., preserve or drop, and for both x_1 and x_2 the property is maintained. We first prove for x_1 with preservation decision:

$$\begin{gathered}\min\{c + f(x_1, x_2 + 1), c' + f(x_1 - 1, x_2 + 1)\} \leq \\ c + f(x_1, x_2 + 1) \leq \\ c + f(x_1 + 1, x_2 + 1)\end{gathered}$$

Similarly we prove for x_1 with drop decision:

$$\begin{gathered}\min\{c + f(x_1, x_2 + 1), c' + f(x_1 - 1, x_2 + 1)\} \leq \\ c' + f(x_1 - 1, x_2 + 1) \leq \\ c' + f(x_1, x_2 + 1)\end{gathered}$$

The proof for x_2 follows similar steps and therefore omitted from the paper. □

Theorem 2 $T_{CP}f(x_1, x_2) : \boldsymbol{Cx(1)} \longrightarrow \boldsymbol{Cx(1)}$ *(convexity in x_1)*

Proof It is important to note that a simpler approach with weaker properties, such as $\mathbf{I} \cap \mathbf{Cx(1)} \rightarrow \mathbf{Cx(1)}$, could have been used to address our specific needs. However, we wish to prove the complete properties of the new preservation operator T_{CP} introduced in this paper.

Similar to [14], we denote the preservation decisions of the last arguments in convexity definition after applying the operator, i.e., $T_{CP}f(X)$ and $T_{CP}f(X + 2e_i)$, as a_1 and a_2 ($a_1 = a_2 = 1$ indicates a decision to preserve a task, and $a_1 = a_2 = 0$ indicates a decision to drop a task). We now examine the four cases:

$\mathbf{a_1 = a_2 = 1}$

We apply convexity once and get:

$$\begin{gathered}2\min\{c + f(x_1 + 1, x_2 + 1), c' + f(x_1, x_2 + 1)\} \leq \\ 2(c + f(x_1 + 1, x_2 + 1)) \leq \\ 2c + f(x_1, x_2 + 1) + f(x_1 + 2, x_2 + 1) \leq \\ c + f(x_1, x_2 + 1) + c + f(x_1 + 2, x_2 + 1)\end{gathered}$$

$\mathbf{a_1 = a_2 = 0}$

Similarly to the previous case, we apply convexity once:

$$\begin{gathered}2\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}\leq\\ 2(c'+f(x_1,x_2+1))\leq\\ 2c'+f(x_1-1,x_2+1)+f(x_1+1,x_2+1)\leq\\ c'+f(x_1-1,x_2+1)+c+f(x_1+1,x_2+1)\end{gathered}$$

$\mathbf{a_1 = 1, a_2 = 0}$

This scenario is straightforward:

$$\begin{gathered}2\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}\leq\\ c+f(x_1+1,x_2+1)+c'+f(x_1,x_2+1)\leq\\ c+f(x_1,x_2+1)+c'+f(x_1+1,x_2+1)\end{gathered}$$

$\mathbf{a_1 = 0, a_2 = 1}$

In this scenario we apply convexity twice:

$$\begin{gathered}2\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}\leq\\ c+f(x_1+1,x_2+1)+c'+f(x_1,x_2+1)\leq\\ c+c'+2f(x_1+1,x_2+1)+f(x_1-1,x_2+1)-f(x_1,x_2+1)\leq\\ c+c'+f(x_1+2,x_2+1)+f(x_1-1,x_2+1)\leq\\ (c'+f(x_1-1,x_2+1))+(c+f(x_1+2,x_2+1))\end{gathered}$$

□

Theorem 3 $T_{CP}f(x_1,x_2)$: ***Super*** $\longrightarrow$ ***Super*** *(supermodularity)*

Proof For clarity we indicate that in the two dimensional state-space **Super(1,2)=Super**. We now prove supermodularity in a similar manner of the convexity proof in Theorem 2, i.e., we examine the four scenarios of a_1 and a_2.

We note that based on [14] (*Equation 6.2*) combining supermodularity and super-convexity properly results in component-wise convexity, i.e., $Super(i,j) \cap SuperC(i,j) \subset Cx(i)$. Super-convexity is proven below in Theorem 7.

$\mathbf{a_1 = a_2 = 1}$

We apply supermodularity once and get:

$$\begin{gathered}\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\ \min\{c+f(x_1,x_2+2),c'+f(x_1-1,x_2+2)\}\leq\\ c+f(x_1+1,x_2+1)+c+f(x_1,x_2+2)\leq\\ c+f(x_1,x_2+1)+c+f(x_1+1,x_2+2)\end{gathered}$$

$\mathbf{a_1 = a_2 = 0}$

We apply supermodularity once and get:

$$\begin{aligned}&\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\&\min\{c+f(x_1,x_2+2),c'+f(x_1-1,x_2+2)\}\leq\\&c'+f(x_1,x_2+1)+c'+f(x_1-1,x_2+2)\leq\\&c'+f(x_1-1,x_2+1)+c'+f(x_1,x_2+2)\end{aligned}$$

$\mathbf{a_1=1,a_2=0}$

This case is straightforward:

$$\begin{aligned}&\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\&\min\{c+f(x_1,x_2+2),c'+f(x_1-1,x_2+2)\}\leq\\&c'+f(x_1,x_2+1)+c+f(x_1,x_2+2)\leq\\&c+f(x_1,x_2+1)+c'+f(x_1+1,x_2+2)\end{aligned}$$

$\mathbf{a_1=0,a_2=1}$

We apply supermodularity twice and get:

$$\begin{aligned}&\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\&\min\{c+f(x_1,x_2+2),c'+f(x_1-1,x_2+2)\}\leq\\&c+f(x_1+1,x_2+1)+c'+f(x_1-1,x_2+2)\leq\\&c+c'+f(x_1,x_2+1)+f(x_1+1,x_2+2)-f(x_1,x_2+2)+\\&f(x_1-1,x_2+1)+f(x_1,x_2+2)-f(x_1,x_2+1)\leq\\&(c'+f(x_1-1,x_2+1))+(c+f(x_1+1,x_2+2))\end{aligned}$$

□

6.2 Additional Properties Maintained by the Preservation Operator

Theorem 4 $T_{CP}f(x_1,x_2):\boldsymbol{UI}\longrightarrow\boldsymbol{UI}$ *(upstream-increase)*

The upstream increase property provides insight on the relations between various dimensions of X. In our use-case X is a two-dimension space and therefore we only need to prove the relation between x_1 and x_2. The proof follows the same technique as in the theorems in Section 6.1 and is detailed in Appendix A.

Theorem 5 $T_{CP}f(x_1,x_2):\boldsymbol{Sub}\cap\boldsymbol{SubC(1,2)}\longrightarrow\boldsymbol{Sub}$ *(submodularity)*

Submodularity property is beneficial in various complex systems and stochastic control problems as detailed in [2]. An example for the use of submodularity in admission-control can be found in [1]. We note that based on [14] (*Equation 6.3*) combining submodularity and sub-convexity properly results in component-wise convexity, i.e., $Sub(i,j)\cap SubC(i,j)\subset Cx(i)$. The proof follows the same technique as in the theorems in Section 6.1 and is detailed in Appendix A.

Theorem 6 $T_{CP}f(x_1,x_2)$: ***SubC(1,2)*** $\longrightarrow$ ***SubC(1,2)*** *(sub-convexity)*

Sub-convexity terminology is from [14] with its definition based on [11], in which it was used in the analysis of arrivals to two queues in series. Similarly to our previous note in Theorem 5, the combination of sub-convexity and submodularity results in convexity ($Sub(i,j) \cap SubC(i,j) \subset Cx(i)$). The proof follows the same technique as in the theorems in Section 6.1 and is detailed in Appendix A.

Theorem 7 $T_{CP}f(x_1,x_2)$: ***SuperC(1,2)*** $\longrightarrow$ ***SuperC(1,2)*** *(super-convexity)*

Similarly to sub-convexity, super-convexity was termed by [14] with the definition based on [11]. We further note that based on [14] (*Equation 6.2*) combining supermodularity and super-convexity properly results in component-wise convexity, i.e., $Super(i,j) \cap SuperC(i,j) \subset Cx(i)$. The proof follows the same technique as in the theorems in Section 6.1 and is detailed in Appendix A.

To conclude, we proved that T_{CP} over a two-dimensional state-space maintains the following properties:

- Non-decreasing: **I** $\longrightarrow$ **I**
- Upstream increase: **UI** $\longrightarrow$ **UI**
- Convexity in one dimension (active servers): **Cx(1)** $\longrightarrow$ **Cx(1)**
- Supermodular: **Super** $\longrightarrow$ **Super**
- Submodular: **Sub** $\cap$ **SubC(1,2)** $\longrightarrow$ **Sub**
- Sub-convexity: **SubC(1,2)** $\longrightarrow$ **SubC(1,2)**
- Super-convexity: **SuperC(1,2)** $\longrightarrow$ **SuperC(1,2)**

7 The Structure of the Admission and Preservation Rule: A Double Switching-Curve

We leverage (9) and use a method presented in [32,33], which relies on a theory presented in [14], to investigate the characteristics of the optimal policy, and show it is of a doubled switching curve type. The method is based on the investigation of the properties of the operators composing the value function (9), and proving that each operator preserves the required properties of the value function.

Theorem 8 *For every fixed value of m_a there are two thresholds level $L_1(m_a)$ and $L_2(m_a)$, such that in state (m_a, m_f) a new task is admitted if and only if $m_f < L_1(m_a)$ and an existing task is preserved if and only if $m_f < L_2(m_a)$.*

The construction of (9) is based on the operators (3)-(8). In Section 6 the properties of the new operator T_{CP} were presented. We remind that it was shown that T_{CP} preserves **I**, **Cx(1)**, and **Super**, in Theorems 1, 2, and 3, respectively. We now focus on the rest of the operators.

The next lemma is required to show the characteristics of operator $T_{D(1,N)}$ in Theorem 9.

Lemma 1 $\overline{\mu}(x) = \min\{x, M\}\mu$ *is nondecreasing and concave in* $x \in \mathbb{N}_0$, $M > 0$.

Proof It is easy to see that $\overline{\mu}(x)$ is nondecreasing in x. We now prove that $\overline{\mu}(x)$ is concave in x, i.e., $2\overline{\mu}(x+1) \geq \overline{\mu}(x) + \overline{\mu}(x+2)$. For $x + 1 > M$ the result follows easily: $2\overline{\mu}(x+1) = 2M\mu = \overline{\mu}(x) + \overline{\mu}(x+2)$. We now address the case of $x + 1 < M$, and we get:

$$2\overline{\mu}(x+1) = 2(x+1)\mu = x\mu + (x+2)\mu = \overline{\mu}(x) + \overline{\mu}(x+2)$$

Finally we address the case where $x + 1 = M$, and we get:

$$2\overline{\mu}(x+1) = 2M\mu > x\mu + M\mu = \overline{\mu}(x) + \overline{\mu}(x+2)$$

□

The following is due to [14] (*Theorem 7.2*, and *Theorem 7.3*). For the proof of $T_{D(1,N)}f(X)$ properties Lemma 1 is required. We note that in [14] *Definition 5.3* $T_{D(1,N)}f(X)$ is defined with normalized rate.

Theorem 9 *([14]* Theorem 7.2 *and* Theorem 7.3*)*

$$T_{A(2)}f(X) : \boldsymbol{I} \to \boldsymbol{I}, \boldsymbol{Cx} \to \boldsymbol{Cx}, \boldsymbol{Super} \to \boldsymbol{Super} \tag{10}$$

$$T_{CA(1)}f(X) : \boldsymbol{I} \to \boldsymbol{I}, \boldsymbol{Cx(1)} \to \boldsymbol{Cx(1)}, \boldsymbol{Super} \to \boldsymbol{Super} \tag{11}$$

$$T_{D1(2)}f(X) : \boldsymbol{I} \to \boldsymbol{I}, \boldsymbol{I(1)} \cap \boldsymbol{Cx} \to \boldsymbol{Cx}, \boldsymbol{Super} \to \boldsymbol{Super} \tag{12}$$

$$T_{D(1,N)}f(X) : \boldsymbol{I} \to \boldsymbol{I}, \boldsymbol{I(1)} \cap \boldsymbol{Cx} \to \boldsymbol{Cx}, \boldsymbol{Super} \to \boldsymbol{Super} \tag{13}$$

The operator T_K that maintains the system within the original finite state-space was introduced in [14]. [1] also uses T_K for similar purposes as ours, i.e., simplifying the investigation of a finite state-space system. The properties of T_K were formally proved in *Lemma 1.5* in [32].

Theorem 10 *([32]* Lemma 1.5*)*

$$T_K f(X) : \boldsymbol{I} \to \boldsymbol{I}, \boldsymbol{Cx} \to \boldsymbol{Cx}, \boldsymbol{Super} \to \boldsymbol{Super} \tag{14}$$

We define a norm on V which will assist in the proof of Lemma 2 and later in the proof of Theorem 8:

$$\|V\| = \max_{(x_1,x_2)} |V(x_1, x_2)| \tag{15}$$

Let S be the set of all functions from $\mathbb{N}_0^2$ to $\mathbb{R}$ that are **I**, **Cx(1)** and **Super**. We now prove that T preserves S, and furthermore T acts as a strict contraction.

Lemma 2 *T and S obey $TS \subset S$. Moreover, there exists a constant $a \in (0, 1)$ such that $\|TU - TW\| \leq a\|U - W\|$ for every $U, W \in S$.*

Proof To prove that $TS \subset S$, let $U \in S$. From Theorem 9, Theorem 10, and Theorems 1,2, and 3 from Section 6, TU being a linear combination of operators which are nondecreasing, convex in x_1, and supermodular, is nondecreasing, convex in x_1, and supermodular. Thus $TU \in S$. As U is arbitrary, it proves that $TS \subset S$.

We note that using the scheme presented in [1] we could use the state dependent event rates, e.g., server failure events, as coefficients in the linear combination of the operators.

To prove the second part of the lemma, we base our proof on [32]. Let $U, W \in S$, we denote $a \vee b = \max(a, b)$ and $a \wedge b = \min(a, b)$ and by applying on (9) the following inequality:

$$|(a \wedge b) - (c \wedge d)| \leq |a - c| \vee |b - d| \tag{16}$$

for $x_1, x_2 \geq 1$ we get:

$$\begin{aligned} &|TU(x_1, x_2) - TW(x_1, x_2)| \\ &\leq \frac{\delta(\lambda + M\mu + \mu_r + 2M\mu_f)}{\Lambda} \|U - W\| \\ &= a\|U - W\| \end{aligned} \tag{17}$$

By definition of δ, $a < 1$. We note that for $x_1 = 0$ and $x_2 = 0$ the calculation is similar and yields the same result. Thus, we conclude that $\|TU - TW\| \leq a\|U - W\|$.

□

Proof (Theorem 8) We use the contraction mapping principle from [30]. The set S (of functions that are nondecreasing, convex in x_1, and supermodular) combined with the metric $\rho(U, W) = \|U - W\|$ is a complete metric space. The map $T : S \to S$ is a strict contraction, as shown in Lemma 2. As a result, T has a unique fixed point. That is, there exists a unique $U \in S$ for which $TU = U$. Recall that V is the unique solution to the same equation in the space of all functions from $\mathbb{N}_0^2$ to $\mathbb{R}$. Therefore, $V = U$ and furthermore $V \in S$, i.e., V is nondecreasing, convex in x_1 and supermodular.

Based on [14], since V is convex then $V(m_a + 1, m_f) - V(m_a, m_f)$ is non decreasing in m_a, and since V is supermodular $V(m_a + 1, m_f) - V(m_a, m_f)$ is nondecreasing in m_f. Thus, the admission of new tasks is nondecreasing in both m_a and m_f, which consequently defines an admission threshold per m_a. Similarly the preservation of existing tasks is nondecreasing in both m_a and m_f, which consequently defines a preservation threshold per m_a. Furthermore, we note that according to [14] the optimal policies are of a switching curve type.

□

To conclude, we showed that both the admission and the preservation rules are of a switching-curve form. In the next section we further analyse the operational rules. The analysis results with a decoupling between the two rules, and in showing that practically one of them is of trivial form. The decoupling result is valid for all cost-structures investigated.

8 Simplifying the Optimal Operational Rule by Decoupling

In this section we investigate the impact of the cost structure on the behavior of the admission and preservation policies and the interplay between them. We will use this analysis to significantly simplify the operational strategy of the system.

We note that based on Section 7 the admission and preservation policies are both of a switching-curve form. Although the general structure of two switching-curve, as illustrated in Figure 3a, seems to be possible, we will show that one of the policies is of a trivial form, i.e., the system will be either in always-preserve or always-admit mode, as illustrated in Figure 3b and Figure 3c, respectively. We will show that the selection between these two rules is based on the relation between rejection (C_r) and drop (C_d) costs. Hence, the influence of the cost structure on the double switching-curve policy is in decoupling the two switching-curves, as one of the switching-curve rules becomes a trivial rule.

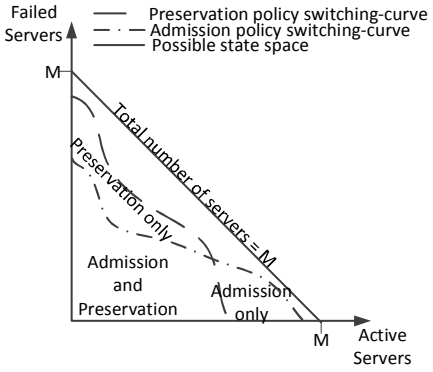
It is important to note that so far the only restriction on cost arguments was $C_a, C_r, C_d \geq 0$. We now assume $C_a < C_r$ which is a necessary condition for the system to admit new tasks, i.e., for the system to be non-empty. Under this assumption, we investigate two types of cost structures $C_d > C_r$ and $C_d < C_r$, and describe the cost regions in which the system conducts admission and/or preservations. One of the outcomes of this investigation, in addition to the policy analysis, is the ability to narrow down the conditions and the relations between the three costs.

We note that Figure 4 depicts the outcome of our analysis. In Section 8.1 we examine the cost region $C_d > C_r$ (regions I, VI, and VII), and in Section 8.2 we examine the cost region $C_d < C_r$ (regions II-V).

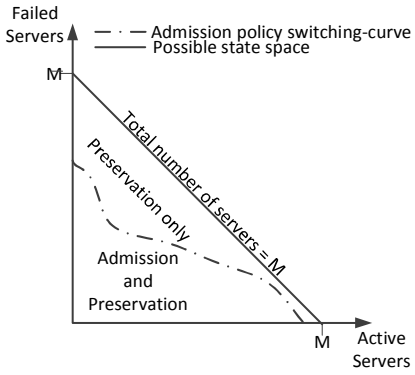
8.1 Rejection is Cheaper than Drop: Never Accept or Always Preserve Systems

In this section we assume $C_d > C_r$, which are regions I, VI, and VII in Figure 4. Under these cost settings dropping an existing task is more costly than rejecting a new task. Furthermore, under these conditions one would expect that the system will be biased towards task preservation and prefer to save idle resources for future preservations rather than utilizing them for gaining immediate revenue due to new arrivals. We will show that in this system, the preservation policy becomes a trivial rule, that is, it is always beneficial to preserve a user (as long as there are available resources), and the preservation switching-curve coincides with $m_a + m_f = M$.

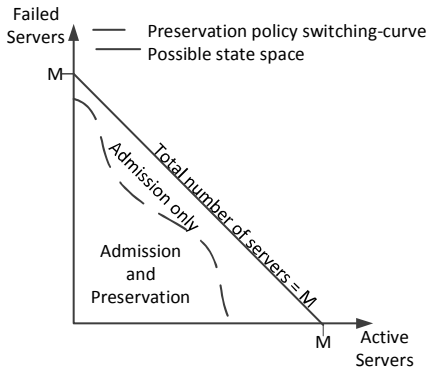
We define $p_f = \mu_f/(\mu + \mu_f)$, which is the probability for a server failure prior to task service completion. We first investigate the conditions for a system to admit tasks, i.e., the system has positive probability to be in states other than $(m_a, m_f) = (0, x)$, where $x = 0, ..., M$ (we term such systems as non-empty systems). We will then use this condition and apply sample path



(a) Two switching curves.



(b) Trivial preservation.



(c) Trivial admission.

Fig. 3: Switching-curves structure.

technique to show that in non-empty systems where the cost parameters obey $C_d > C_r$ the preservation rule is of a trivial form, i.e., the rule is always-preserve.

Lemma 3 (Regions VI, and VII) *If $C_r < C_a/(1-p_f)$ then the system will never accept a new task, i.e., the system is an empty system.*

Proof The proof is based on the analysis of potential events upon a task admission and comparison of the overall cost of the event sequences to the cost of immediate rejection (C_r).

We first examine a system with an infinite number of servers. Although this is not a realistic situation in cloud computing, we conduct this analysis for the sake of the proof completeness. We note that with infinite number of servers there is no concern to drop a task. An admitted task will be continuously maintained until the service is completed. Therefore, the overall cost the system invests in server assignment (C_a) until task completion is:

$$C_a + p_f(C_a + p_f(...)) = C_a \sum_{i=0}^{\infty} (p_f)^i = \frac{C_a}{1-p_f} \tag{18}$$

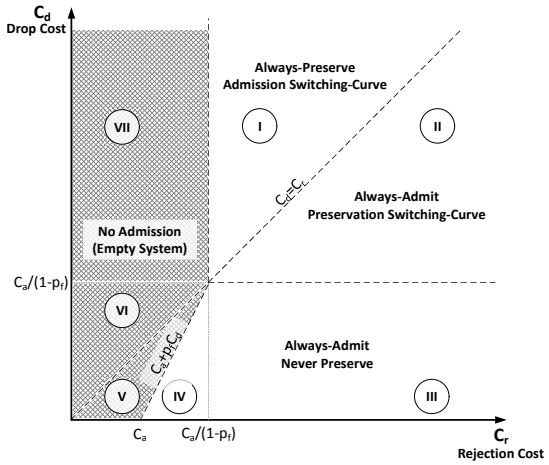


Fig. 4: The policy behavior as a function of the cost structure.

Hence, in a system with infinite number of servers, if $C_r < C_a/(1-p_f)$ tasks will not be admitted and the system is an empty-system.

We will now focus on systems with finite number of servers. In such systems there is a positive probability that after certain number of preservations the system will be required to drop the task and incur the C_d penalty. We will show that for any sequence of preservations followed by a drop event, under the aforementioned condition the system will not accept any task.

Given that the drop of the customer occurs on the $(k+1)st$ failure the operation cost is:

$$C_a \sum_{i=0}^{k} (p_f)^i + (p_f)^{k+1} C_d, k \geq 0 \tag{19}$$

We note that $C_d > C_r$ and $C_r < C_a/(1-p_f)$ (the lemma's condition), and substitute them in (19) to yield:

$$C_a \sum_{i=0}^{k} (p_f)^i + (p_f)^{k+1} C_d > (1-p_f) C_r \sum_{i=0}^{k} (p_f)^i + (p_f)^{k+1} C_r = C_r \tag{20}$$

We emphasize that (20) holds for all $k \geq 0$. Hence, as the overall operational cost for all potential preservations and drop sequences is higher than that of rejecting the task upon arrival, the system will never accept a new task.

□

We conclude that $C_r \geq C_a/(1-p_f)$ is a necessary condition for admitting a task in systems with either infinite or finite number of servers.

We now focus on systems where $C_d > C_r$ and where the system is non-empty (that is, $C_r \geq C_a/(1-p_f)$), and investigate the preservation decision. We prove that under this cost structure it is always beneficial to conduct task

preservation, i.e., the preservation policy if of a trivial rule. For clarity we note that the following analysis considers a system with a finite number of servers.

Theorem 11 (Region I) *Consider a system with finite number of servers and* $C_r \geq C_a/(1-p_f)$ *and* $C_d > C_r$. *Then for all* $(x_1, x_2), x_1 \geq 1, x_2 \geq 0$ *such that* $x_1 + x_2 < M$ *it is optimal to preserve an existing task, i.e.,* $C_a + V(x_1, x_2+1) \leq C_d + V(x_1-1, x_2+1)$.

Proof We will prove it by a way of contradiction. For the sake of contradiction assume that the optimal policy is policy B, that in some cases does not obey the preservation rule of the theorem. Let us look at a system applying policy B (system B) and at the first such event in which system B violates the rule, that is it does not preserve a task though it has an idle server. Assume this happens at t_0.

We will construct a policy A that will perform better than policy B. System A applying policy A is identical to system B until t_0 in which it continues to preserve the task dropped by system B. We denote this task as the *extra task*, and the rest of the tasks (if exist) that are in both systems as the *mutual tasks*. Using a sample-path technique, we couple the two systems via the failure, service and arrival times, i.e., all server failures, task departures and arrivals are the same in both systems. Once system B drops the task the value function difference between the systems is $C_d - C_a$. We note that from time t_0 system A precisely imitates system B, until at some point in time, say t_1:

1. A new task arrives and system A needs to reject it (lack of idle servers), but system B can accept the task (due to the additional idle server). Thus in this case the cost difference between the systems is $C_d - C_r$. We note that from t_1 on system A imitates system B (both systems will behave similarly).
2. An active server of one of the mutual tasks fails and system A drops the task, but system B can switch to another server (due to the extra idle server). In this case the cost difference between the two system is 0. Also here from t_1 on both systems will behave similarly.
3. The extra task in system A is completed. In this case the cost difference between the systems remains $C_d - C_a$, and from this point (t_1) on both systems will behave similarly.
4. The server of the extra task in system A fails. System A then decides to drop the task (the delayed drop as mentioned above), which results in a penalty of C_d. In system B this server is idle and therefore system B does not incur any additional cost. Also here from this point (t_1) on both systems will behave similarly.

We emphasize that the two systems are completely coupled before t_0, and then any time between t_0 and t_1, and then after t_1. Therefore, we cover all trajectories at which the systems differ from each other.

We note that each of the aforementioned scenarios occurs with a certain probability which we must account for. Due to the cost structure, it is obvious that $C_d - C_a > 0$ and $C_d - C_r > 0$. We focus on scenarios 3 and 4, both of

the scenarios depend on a single task (the extra task) behavior. Therefore, the probabilities of scenarios 3 and 4 are $1 - p_f$ and p_f (p_f defined in Lemma 3), respectively. We now evaluate the cost difference between system A and system B. It is obvious that system B overall expected cost is $C_B = C_d$. System A overall expected cost is $C_A = C_a + p_f C_d$. We compare the two costs (C_A and C_B):

$$C_A - C_B = C_a + p_f C_d - C_d = C_a - (1 - p_f)C_d \tag{21}$$

Based on the conditions of the theorem, i.e., $C_r \geq C_a/(1 - p_f)$, and the fact that $C_d > C_r$, it is clear from (21) that $C_A < C_B$. Thus, system A achieves an overall lower expected system cost than system B, considering all scenarios and their occurrence probabilities. We note that as our investigation is of an arbitrary task at an arbitrary point in time (and hence in system state) we can conclude that it is always beneficial to preserve an already admitted task (as long as there are available servers) when an active server fails.

□

Corollary 1 (Region I) *Under the cost structure $C_d > C_r > C_a/(1 - p_f)$, the system admits new tasks, i.e., the system is non-empty, and always prefers to preserve an existing task.*

Corollary 2 (Regions VI and VII) *Under the cost structure $C_d > C_r$ and $C_r < C_a/(1 - p_f)$, it is not beneficial for the system to conduct admissions and therefore the system is empty, i.e., the admission rule is of trivial form and coincides with $m_f = 0$.*

8.2 Drop is Cheaper than Rejection ($C_d < C_r$): Always Admit Systems

In this section we assume $C_d < C_r$, corresponding to regions II-V in Figure 4. To this end we note that admission and preservation are important in various applications. In some of these applications, one may expect rejections to be more expensive than drops; For example in cellular networks if the operator gives priority to admission of new incoming calls versus the handling of handover calls, or in certain service centers (such as hospitals) in which the admission of a new user is more important than preserving users that are already in the systems. We note that systems who operate this way are sometimes called *overnight operation*. While we are not familiar with such setting in cloud computing we conduct the analysis for the sake of completeness.

We maintain the definitions of p_f from Section 8.1. We apply a technique similar to that used in the pervious section, yet we first investigate the conditions for preservation, and only afterwards investigate the conditions for task admission, and prove the always-admit policy.

8.2.1 The Preservation Policies: Never Preserve when $C_d < C_a/(1-p_f)$

We consider a task that is present in the system[1] and investigate the conditions under which task preservation is not performed. Task preservation impacts the operational cost through a sequence of preservations and possibly a final drop. Therefore the analysis is similar to that of Lemma 3.

Lemma 4 (Regions III-V) *If $C_d < C_a/(1-p_f)$ then the system will never conduct task preservations.*

Proof The proof relies on the fact that the expected cost on preserving a task is more costly than dropping, and is similar to the proof of Lemma 3. The detailed proof is given in Appendix B.

We note that the aforementioned condition divides the cost structure to two regions. When $C_d < C_a/(1-p_f)$ (regions III-V), it is not beneficial to conduct preservation at all, and therefore all admitted tasks will be dropped once their server fails. Hence, the preservation policy when $C_d < C_a/(1-p_f)$ is of a trivial form: $m_f = 0, \forall m_a \geq 0$.

Note that We will not show special preservation rules for the case $C_d \geq C_a/(1-p_f)$. In this case preservation will follow the general structure (switching curve) proved in Section 7.

8.2.2 Admission Policy: Never Admit (Region V) or Always Admit (Regions II-IV).

In this section we investigate the behavior of the admission policy. We first investigate the admission conditions when $C_r > C_d$, namely the system is biased towards new admissions. We then prove that under these conditions the admission switching curve is of a trivial form, i.e., the system applies either an always admit or an always reject policy. For clarity we note that the analysis is based on a system with finite number of servers.

Theorem 12 (Region V) *If $C_d < C_a/(1-p_f)$ and $C_r < C_a + p_f C_d$ then the system will never accept new tasks.*

Proof Based on Lemma 4, when $C_d < C_a/(1-p_f)$ a task will be dropped once it's server fails. Therefore, the overall operational cost considering the potential drop is:

$$C_a + p_f C_d. \tag{22}$$

Hence, if $C_a + p_f C_d > C_r$ it is not beneficial to admit the task and it will be dropped.

□

We now investigate the preservation region $C_a/(1-p_f) > C_d$, in which the system does not conduct preservation.

[1] We do not deal yet with whether the system decided to admit it.

Theorem 13 (Regions III and IV) *If $C_r > C_d$, $C_r > C_a + p_f C_d$ and $C_a/(1-p_f) > C_d$ then for all $(x_1, x_2), x_1 \geq 1, x_2 \geq 0$, such that $x_1 + x_2 < M$, it is optimal to admit a new task, i.e., $C_a + V(x_1 + 1, x_2) \leq C_r + V(x_1, x_2)$.*

Proof In this region an admitted task is dropped once its server fails (never preserve region). Nonetheless, since $C_r > C_a + p_f C_d$, even accounting for the potential drop cost the admission is beneficial. Hence, the system will always conduct admissions.

□

We continue and examine the admission policy. We will show that in this cost structure ($C_r > C_d$) it is always beneficial to admit new tasks. We will now examine system in which preservation may possibly be conducted, i.e., $C_d \geq C_a/(1-p_f)$ (region II). We note that the following theorem somewhat resembles Theorem 11, yet the different cost structure requires a completely new analysis.

Theorem 14 (Region II) *If $C_r > C_d$ and $C_d \geq C_a/(1-p_f)$ then for all $(x_1, x_2), x_1 \geq 1, x_2 \geq 0$, such that $x_1 + x_2 < M$, it is optimal to admit an existing task, i.e., $C_a + V(x_1 + 1, x_2) \leq C_r + V(x_1, x_2)$.*

Proof We note that the proof somewhat resembles that of Theorem 11 and involves an analysis based on sample-path; Yet the different cost structure requires a completely new analysis. The proof is detailed in Appendix C.

□

We conclude that a system in a cost region where $C_r > C_d$, i.e., regions II-IV, will conduct an always-admit policy and will prefer the admission of new tasks versus preservation of existing tasks.

8.3 The Effect of the Cost Parameters on Admissions and Preservations: Discussion

In Section 7 we showed that the admission and preservation rules are of a switching-curve form. Through our analysis in Section 8.1 and Section 8.2 we were able to eliminate one of the operational rules, as we proved it to be of trivial form. Recall that Figure 4 illustrates a summary of the admission and preservation policies as a function of the cost parameters C_r and C_d, given the server allocation cost C_a and server failure probability p_f. The line $C_r = C_d$ divides the cost space into two main regions: $C_d > C_r$ and $C_d < C_r$, in which the system is biased towards preservations and admissions, respectively. Within the region $C_d > C_r$, if the system is not-empty, then the system conducts an always-preserve policy (region I). For the region $C_d < C_r$, the line $C_d = C_a/(1-p_f)$ divides the region into two sub-regions in which the preservation is of a switching-curve type ($C_d \geq C_a/(1-p_f)$) in region II, or in the form of a trivial never-preserve rule ($C_d < C_a/(1-p_f)$) in regions III and IV. As shown in both Lemma 3 and Theorem 12 there are regions in which

the system is an empty system regardless of the preservation behavior: regions V-VII. We emphasize that the actual admission threshold depends, in addition to the cost structure, also on the various parameters of the system, e.g., task arrival rate and server replacement rate.

We note, that when $C_d = C_r$ and $C_r, C_d \geq C_a/(1-p_f)$ (the border line between regions I and II), then both admission and preservation are of a trivial form, i.e., always admit and always preserve.

8.4 Does Preemption Pay Off?

We now consider a system where task preemption is allowed, i.e., if a server is required for admission of a new task and there are no idle servers then the system is allowed to dynamically select an existing task and drop it (while incurring the required cost) in order to allocate the server to the new task. We investigate the impact of preemption on systems with $C_r > C_d$. In these system settings, as long as it beneficial to conduct preservations, i.e., $C_a/(1-p_f) \leq C_d$, then the optimal preservation rule is of a trivial form. Meaning, the system will prefer to conduct preservations in all state-space (identical to the admission policy). The reason is that as long as there is no need for the idle servers the system benefits from conducting preservations instead of being penalized with drop costs. Yet, once a new task arrives and requires resources, the system can select one of the existing active tasks and drop it to free one of the servers for the new task (while incurring the drop cost). As $C_r > C_d$ this behavior is beneficial for the overall system cost. Therefore, if preemption is allowed and $C_r > C_d$ then both admission and preservation rules are the trivial rules: for all $(x_1, x_2), x_1 \geq 1, x_2 \geq 0$, such that $x_1 + x_2 < M$ always conduct admission and preservation.

We note that in the system described in Section 8.1, i.e., where $C_d > C_r$, preemption cannot be employed as it is not cost beneficial (requires payment of C_d). Therefore, since there are no preemptions, the analysis and results presented in Section 8.1 hold.

9 Behavior of Admission and Preservation: Numerical Results

In this section we use numerical results to investigate the impact of the various system parameters, for each of the cost structures described in Section 8, on the admission and preservation switching-curve policies.

We examine a system with $M = 50$ servers. We will study the system under two types of tasks; in one setting we will consider short tasks with $1/\mu = 5$ and $\lambda = 40$; and in the other long tasks with $1/\mu = 20$ and $\lambda = 10$. Server replacement rate is $\mu_r = 1/60$. We consider two types of cloud systems[2]: one where the servers are prone to failure with $1/\mu_f = 30$, and one where they are semi-reliable with $1/\mu_f = 120$. Thus, our investigation includes four types of

[2] We use $1/\mu_f = 30, 120$ to emphasize the impact of the properties under investigation.

systems with variety of tasks and servers configurations. In our runs we use a discount factor $\delta = 0.9999$[3], and convergence criterion of relative error 0.1%[4]. Table 1 details the server failure probability p_f (defined in Section 8) for the four systems.

Table 1: p_f of the four systems.

	Semi-Reliable	Prone-to-Failures
Short Tasks	0.143	0.4
Long Tasks	0.04	0.143

We evaluate the four systems with two cost structures (C_d, C_r): $(100, 50)$ and $(50, 100)$. Meaning one system with $C_d > C_r$ and another with $C_r > C_d$. In addition, we set $C_a = 1$. The first cost structure is more realistic and very intuitive when considering cloud computing systems. The latter cost structure is interesting to investigate as it incorporates a non intuitive system behavior.

Figure 5a depicts the policy for the four systems under $C_d > C_r$; recall that under these conditions the policy is of a switching curve form. We now examine the impact of the failure and service rates on the switching-curve behavior:

- Failure rate (μ_f): the admission policy is more conservative when failure rate increases. As can be seen in Figure 5a when comparing between semi-reliable and prone-to-failure systems. This is mainly due to the fact that there is higher probability that a server will fail during an active task.
- Service rate (μ): the admission decision is more conservative when the system needs to handle long tasks, i.e., when the service rate is low. This is due to the fact that task lifetime in the system has significant impact on the probability of an active server failure (as can also be seen in Table 1) which then translates into a preservation decision and potentially dropping penalty.

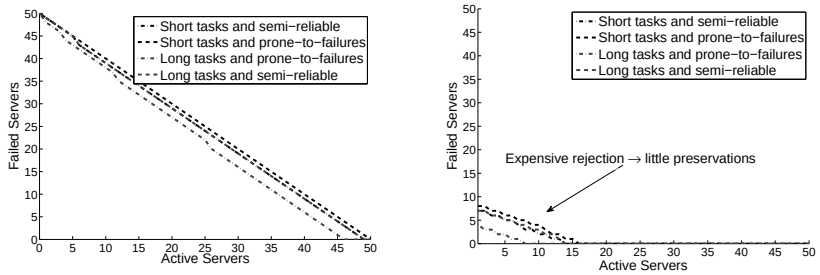
Figure 6 depicts the switching curves of a system with long tasks and prone-to-failure servers with three drop costs: $200, 100$ and 75 ($C_r = 50$ and $C_a = 1$). We explain the impact of the drop cost on the preservation policy behavior:

- Drop cost (C_d): system admission (admission region) decreases with drop-costs. That is, when the drop costs increase the system must be more conservative in accepting new tasks, as can beseen in Figure 6.

Figure 5b illustrates the switching curves of the four system with $C_r > C_d$. We first note that the switching-curves in this settings are more conservative compared to the ones in the four systems with $C_d > C_r$. The reason for this

[3] We note that δ is set to this very high value as our system includes virtual events (due to the uniformization), that do not impact the value function yet force us to give more weight to actual events.

[4] We stop the value iteration once the relative change of V (for all elements in the state-space) between consecutive iterations drops below the convergence criterion.



(a) Admission switching-curves when $C_d > C_r$.

(b) Preservation switching-curves when $C_d < C_r$.

Fig. 5: Switching curves of the four systems.

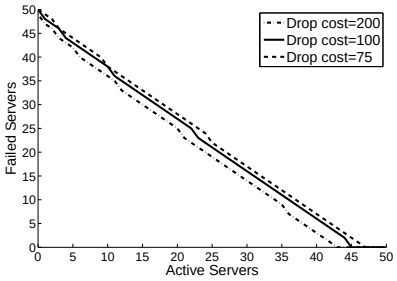


Fig. 6: Switching curves of several drop costs for a system with long tasks and prone-to-failures servers ($C_d > C_r$).

behavior is the fact that new arrivals (of tasks) rate is higher than active servers failures. Therefore, to prevent high penalties, the system needs to maintain more idle servers (compared to the system with $C_d > C_r$) for future arrivals. We now examine the impact of the failure and service rates on the switching-curve behavior:

- Failure rate (μ_f): the preservation policy is more conservative when failure rate increases. As can be seen in Figure 5b when comparing between semi-reliable and prone-to-failure systems. This is mainly due to the fact that there is higher probability that a server will fail during an active task, which will be needed to conduct additional preservations.
- Service rate (μ): the preservation decision is more aggressive when the system needs to handle short tasks. This can be seen in Figure 5b when comparing the semi-reliable systems. We note that prone-to-failures systems behave similarly due to the impact of the arrival rate. For short tasks the arrival rate is high, which results in a conservative policy.

10 Heuristic Approximation of Large Systems

In this section we propose two heuristic approaches to overcome the computational efforts in analyzing a large state-space. We focus on systems with $C_d > C_r > C_a/(1-p_f)$, a cost structure that is most relevant to CCC, and leverage the proposed schemes to derive the admission policy (as the preservation policy is of a trivial form). We note that these heuristic approaches reduce the run time of the value-function calculation, as well as the computation efforts in extracting the admission switching-curve policy. As these heuristic approaches achieve a sub-optimal policy, we evaluate their performance using numerical results and compare the achieved admission switching-curve with the optimal curve. Furthermore, we investigate the expected operational cost of a system with the sub-optimal policies by applying the approximated switching curve into the value-function calculation. Based on this method we can then derive the expected additional operational cost due to the use of a sub-optimal switching-curve. In addition we provide the conditions on the system settings in which each of the approaches can be applied.

10.1 State-Space Reduction Heuristic

The first heuristic is based on the reduction of the state space by a factor F, e.g., reduction from 500 servers to 50 servers means $F = 10$. The interpretation to this reduction is that every server in the reduced state-space reflects F servers in the original state-space. Due to the change in the state-space, other system settings are affected directly or indirectly. Task completion and server failure event rates do not change but are effectively reduced as they are state dependent. Task arrival rate and server replacement rate are required to be reduced by the same factor F. Once we derive the switching-curve in the new system, we can then linearly extrapolate it back to the original state-space. We note that the accuracy of this method decreases as the importance of servers granularity increases, for example when the active number of servers is closer to the total number of servers (as will be explained below). Furthermore, when $\mu_f < \mu$ the accuracy of the approximation is affected as well, as the importance of each idle server increases.

10.2 Single Task Weight Heuristic

The second heuristic is based on the direct cost derived by a state-transition, e.g., admitting/adding a task. We note that due to this approach, the suggested heuristic cannot be applied to the analysis of complex system behavior and therefore is applicable to system settings were $\mu_f < \mu$. In such systems, an admitted task, on average, utilizes less than two servers during service time. As a result, the direct impact of a single task on the operational cost can be leveraged to estimate the total system cost. Furthermore, by leveraging the

assumption on the linearity of the switching curve (a reasonable assumption based on observation of hundreds of systems), we can then focus only on the investigation of the intersection points with the two axes. The combination of the aforementioned factors, linear switching-curve, focus on two intersection points, and single task analysis, enables us to significantly reduce the system analysis complexity. We emphasize that with this heuristic approach, we analyze the maximal immediate cost, i.e., the fastest event sequence to reach either a drop penalty or a reject penalty due to the single task.

We now investigate the intersection points with the two axes. The intersection point with the x-axis is extracted by finding the minimum task's operational cost over the possible states, i.e., $\min_{i=1,\ldots,M}\{C_{drop}(i,0)+C_{reject}(i,0)\}$. Where $C_{drop}(i,j)$ (defined in (23)) and $C_{reject}(i,j)$ (defined in (24)) are the drop and rejection costs projected from the state with i active servers and j failed servers.

$$C_{drop}(i,j) = \quad (C_d + (M-i-1)C_v) \prod_{k=j}^{M-i} \frac{i\mu_f}{\Omega(k,i)} \tag{23}$$

$$C_{reject}(i,j) = C_r \sum_{k=j}^{M-i-1} \frac{\lambda}{\Omega(k,i)} \left(\frac{i\mu_f}{\Omega(k,i)}\right)^{k-j} \tag{24}$$

where, $\Omega(k,i) = (M-k)\mu_f + i\mu + \lambda + \mu_r$.

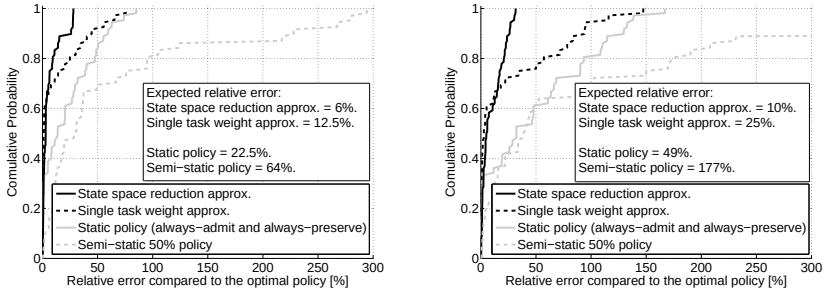
The intersection point with the y-axis is calculated by comparing the cost of immediate rejection when there are no active servers, i.e., $C_r\frac{\lambda}{\lambda+\mu}$, to the operational cost over the possible number of failed servers and single active server. Meaning, $\max_{i=0,\ldots,M-1}\{C_{drop}(1,i) + C_{reject}(1,i)\}$, s.t. $C_{drop}(1,i) + C_{reject}(1,i) < C_r\frac{\lambda}{\lambda+\mu}$.

10.3 Quality of the Approximation Methods - Numerical Results

We will evaluate the approximation methods through numerical investigation in variety of system settings, and show they achieve good results compared to the optimal policy. We use the comparison method from [21], and compare the value function at the origin ($V(0,0)$) of the systems to the one of the optimal switching-curve. In addition, the comparison includes two additional policies: one with a simple policy of admitting and preserving tasks as long as there are available idle servers (i.e., always-admit and always-preserve static policy), and another one (a semi-static policy) which admits new tasks as long as 50% of the non-active servers are idle.

We examine the methods over a wide set of cases (216 cases outlined below) and summarize the method accuracy by providing a histogram of their errors. The histograms for the four schemes are presented in Figures 7a and Figure 7b, in the form of a cumulative distribution function (CDF), for the cost structures $(C_d, C_r, C_a) = (50, 10, 1)$ and $(C_d, C_r, C_a) = (500, 50, 1)$, respectively.

The various cases we examine constructed by varying the parameters as follows: $M = 500$, $\lambda = 10, 25, 50, 100$, $\mu_r = 1/30, 1/60, 1/150$, $\mu = 1, 1/15, 1/30$, $\mu_f = 1/10, 1/30, 1/60$, and $\delta = 0.9999$ (we apply the same analysis method as described in Section 9).



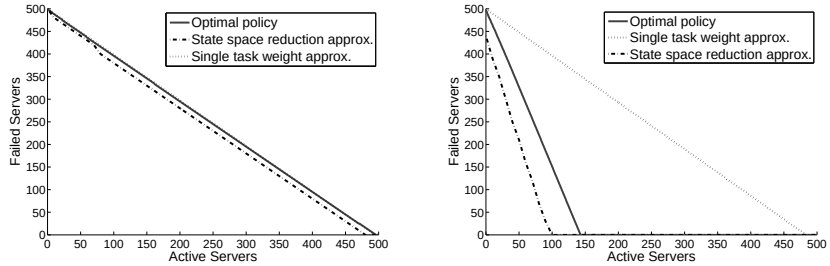
(a) $(C_d, C_r, C_a) = (50, 10, 1)$. (b) $(C_d, C_r, C_a) = (500, 50, 1)$.

Fig. 7: Cumulative distribution functions of the relative error.

The results demonstrate that the state-space reduction approximation performs well with relatively small errors and outperforms all other methods throughout all the scenarios. The single task weight approximation performs significantly better than the static and semi-static policies but in some cases it is significantly less accurate than the state-space reduction method. The latter inaccuracy results from the scheme's aggressive switching-curve in scenarios where $\mu < \mu_f$, as explained below. Comparing between the two cost structures, it can be seen that when the difference between C_d and C_r increases the static and semi-static policies errors increase significantly. The static policy may reach high error levels of above 100%. The semi-static policy behaves even worse, and reaches error levels above 200%, which means 3 times more costly than the optimal policy. While the two approximation schemes significantly outperform the static and semi-static policies, it can be seen that the state-space reduction approximation scheme is better than the single-task weight approximation scheme.

Figure 8 compares the admission rule (the switching curve) of the approximation methods to the optimal one. In Figure 8a we examine the case $\mu \geq \mu_f$ and it can be seen that both approximations closely follow the optimal rule. Specifically, for the state-space reduction approximation, the difference from the optimal policy slightly increases as the number of active server increases. This is due to the fact that as more servers are active the importance of the remaining number of idle servers increases and, thus, in this region the fine-grained information on servers quantity is significant. On the other hand, in the case where $\mu < \mu_f$ (studied in Figure 8b) the two approximations are less effective. This is due to the fact that the approximations cannot mimic the complex behavior of the system and thus derive a switching-curve which is

either too conservative (state space reduction) or too aggressive (single task weight).



(a) Practical failure rates ($\mu \geq \mu_f$) - very good approximation.

(b) Extremely high failure rates ($\mu < \mu_f$) - bad approximation.

Fig. 8: Switching-curve approximation for $\lambda = 10$, $\mu = 1/30$, $\mu_f = 1/10, 1/60$, $\mu_r = 1/30$, $M = 500$ and $(C_d, C_r, C_a) = (50, 25, 1)$.

We conclude and emphasize that in complex settings it is necessary to evaluate the admission policy switching-curve through the full MDP formulation. While in scenarios where $\mu \geq \mu_f$ our two approximation schemes perform well with negligible difference.

11 Concluding Remarks, Applications and Future Work

This paper presented a new problem formulation of task admissions and preservations in CCC using a queueing-loss multi-server system with failures. We used an MDP, constructed a new preservation operator and derived the optimal task and admission policy, showing that it consists of a double switching curve. Our analysis revealed that the overall solution can be simplified and the decisions can be virtually decomposed. Based on that, two heuristic approaches were introduced which further simplify the extraction of the operational rule.

The proposed task management scheme can be applied in traditional CCC, where tens of thousands of servers are located in a centralized location, as well as in distributed computing paradigms such as Distributed Cloud Computing (DCC) [5] and potential future Mobile Cloud Computing (MCC) [4, 31]. In DCC our scheme can be applied to manage the tasks as in traditional CCC and enable the DCC manager to leverage lower cost (and more prone to failures) computing platforms. In MCC our scheme can be applied to overcome the dynamic nature of the mobile devices that can leave and join the group/cluster. A mobile leaving the group can be modeled as a failed server.

Throughout the paper we focused on resource and task management scheme for the CCC. Yet our scheme can also be applied by Service-Providers (SP's) renting resources (servers/VMs) from the CCC under a non-guaranteed Service Level Agreement (SLA). In this type of SLA, allocated resources may be

reclaimed by the CCC for other needs, e.g., serving SP's under guaranteed SLA. As a result, SP's under a non-guaranteed SLA are operating in a dynamic environment in which servers may be taken out of their service, which is equivalent to a failed server in our system model. Thus, our scheme can be used by SP's to dynamically manage their rented servers pool and ensure continuous service in spite of their server recapturing by the CCC.

Throughout this paper we focused on a single type of tasks and identical servers. A potential future research direction is to extend the scheme to support a heterogeneous computing environment where either the computing resources have different capabilities or tasks have different priorities, cost structure, or reliability demand. We note that as a possible sub-optimal solution, one may employ the scheme proposed in this paper in a per platform manner.

We conclude that our proposed scheme provides a holistic task management solution for the complex cloud computing environment. The newly composed preservation operator introduces a new tool for the analysis of queueing-systems, and its various properties are detailed and can be employed in other types of systems.

References

1. Altman, E., Jimenez, T., Koole, G.: On optimal call admission control in resource-sharing system. Communications, IEEE Transactions on **49**(9), 1659–1668 (2001). DOI 10.1109/26.950352
2. Altman, E., Koole, G.: On submodular value functions and complex dynamic programming. Communications in Statistics. Stochastic Models **14**(5), 1051–1072 (1998). DOI 10.1080/15326349808807514
3. Bin, E., Biran, O., Boni, O., Hadad, E., Kolodner, E., Moatti, Y., Lorenz, D.: Guaranteeing high availability goals for virtual machine placement. In: Distributed Computing Systems (ICDCS), 2011 31st International Conference on, pp. 700–709 (2011). DOI 10.1109/ICDCS.2011.72
4. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC '12, pp. 13–16. ACM, New York, NY, USA (2012). DOI 10.1145/2342509.2342513. URL http://doi.acm.org/10.1145/2342509.2342513
5. Chandra, A., Weissman, J., Heintz, B.: Decentralized edge clouds. Internet Computing, IEEE **17**(5), 70–73 (2013). DOI 10.1109/MIC.2013.93
6. Chen, X., Lu, C.D., Pattabiraman, K.: Failure prediction of jobs in compute clouds: A google cluster case study. In: Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on, pp. 341–346 (2014). DOI 10.1109/ISSREW.2014.105
7. Dean, J.: Designs, lessons and advice from building large distributed systems (2009). URL https://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf. Keynote Speech at the 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware
8. Efrosinin, D.: Queueing model of a hybrid channel with faster link subject to partial and complete failures. Annals of Operations Research **202**(1), 75–102 (2013). DOI 10.1007/s10479-011-0939-7
9. Feldman, Z., Masin, M., Tantawi, A., Arroyo, D., Steinder, M.: Using approximate dynamic programming to optimize admission control in cloud computing environment. In: Simulation Conference (WSC), Proceedings of the 2011 Winter, pp. 3153–3164 (2011). DOI 10.1109/WSC.2011.6148014

10. Garraghan, P., Townend, P., Xu, J.: An empirical failure-analysis of a large-scale cloud computing environment. In: High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on, pp. 113–120 (2014). DOI 10.1109/HASE.2014.24
11. Ghoneim, H.A., Jr., S.S.: Control of arrivals to two queues in series. European Journal of Operational Research **21**(3), 399 – 409 (1985). DOI http://dx.doi.org/10.1016/0377-2217(85)90160-2
12. Hu, Z., Wu, K., Huang, J.: An utility-based job scheduling algorithm for current computing cloud considering reliability factor. In: Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference on, pp. 296–299 (2012). DOI 10.1109/ICSESS.2012.6269464
13. Javadi, B., Abawajy, J., Buyya, R.: Failure-aware resource provisioning for hybrid cloud infrastructure. J. Parallel Distrib. Comput. **72**(10), 1318–1331 (2012). DOI 10.1016/j.jpdc.2012.06.012
14. Koole, G.: Monotonicity in markov reward and decision chains: Theory and applications. Foundations and Trends in Stochastic Systems **1**(1), 1–76 (2006). DOI 10.1561/0900000002. URL http://dx.doi.org/10.1561/0900000002
15. Lerzan Örmeci, E., Burnetas, A., van der Wal, J.: Admission policies for a two class loss system. Stochastic Models **17**(4), 513–539 (2001). DOI 10.1081/STM-120001221
16. Lewis, M.E., Ayhan, H., Foley, R.D.: Bias optimality in a queue with admission control. Probability in the Engineering and Informational Sciences **13**, 309–327 (1999)
17. Limrungsi, N., Zhao, J., Xiang, Y., Lan, T., Huang, H., Subramaniam, S.: Providing reliability as an elastic service in cloud computing. In: Communications (ICC), 2012 IEEE International Conference on, pp. 2912–2917 (2012). DOI 10.1109/ICC.2012.6364649
18. Machida, F., Kawato, M., Maeno, Y.: Redundant virtual machine placement for fault-tolerant consolidated server clusters. In: Network Operations and Management Symposium (NOMS), 2010 IEEE, pp. 32–39 (2010). DOI 10.1109/NOMS.2010.5488431
19. Malik, S., Huet, F.: Adaptive fault tolerance in real time cloud computing. In: Services (SERVICES), 2011 IEEE World Congress on, pp. 280–287 (2011). DOI 10.1109/SERVICES.2011.108
20. Malik, S., Huet, F., Caromel, D.: Reliability aware scheduling in cloud computing. In: Internet Technology And Secured Transactions, 2012 International Conference for, pp. 194–200 (2012)
21. Milito, R., Levy, H.: Modeling and dynamic scheduling of a queueing system with blocking and starvation. Communications, IEEE Transactions on **37**(12), 1318–1329 (1989). DOI 10.1109/26.44203
22. Miller, B.L.: A queueing reward system with several customer classes. Management Science **16**(3), 234–245 (1969). DOI 10.1287/mnsc.16.3.234
23. Mitrany, I.L., Avi-Itzhak, B.: A many-server queue with service interruptions. Operations Research **16**(3), 628–638 (1968). DOI 10.1287/opre.16.3.628
24. Mondal, S., Muppala, J., Machida, F., Trivedi, K.: Computing defects per million in cloud caused by virtual machine failures with replication. In: Dependable Computing (PRDC), 2014 IEEE 20th Pacific Rim International Symposium on, pp. 161–168 (2014). DOI 10.1109/PRDC.2014.29
25. Neuts, M.F., Lucantoni, D.M.: A markovian queue with n servers subject to breakdowns and repairs. Management Science **25**(9), 849–861 (1979). DOI 10.1287/mnsc.25.9.849
26. Ni, J., Tsang, D., Tatikonda, S., Bensaou, B.: Threshold and reservation based call admission control policies for multiservice resource-sharing systems. In: INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE, vol. 2, pp. 773–783 vol. 2 (2005). DOI 10.1109/INFCOM.2005.1498309
27. Özkan, E., Kharoufeh, J.: Incompleteness of results for the slow-server problem with an unreliable fast server. Annals of Operations Research pp. 1–5 (2014). DOI 10.1007/s10479-014-1615-5
28. Özkan, E., Kharoufeh, J.P.: Optimal control of a two-server queueing system with failures. Probability in the Engineering and Informational Sciences **28**, 489–527 (2014). DOI 10.1017/S0269964814000114
29. Puterman, M.L.: Markov decision processes : discrete stochastic dynamic programming. Wiley series in probability and mathematical statistics. John Wiley & Sons, New York (1994). URL http://opac.inria.fr/record=b1084090. A Wiley-Interscience publication.

30. Reed, M., Simon, B.: Methods of Modern Mathematical Physics Vol 1: Functional Analysis, 1 edn. Academic Press (1981)
31. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for vm-based cloudlets in mobile computing. Pervasive Computing, IEEE **8**(4), 14–23 (2009). DOI 10.1109/MPRV.2009.82
32. Shifrin, M.: Admission and scheduling control in cloud computing - markov decision processes and difusion approximations. Ph.D. thesis, Technion - Israel Institute of Technology (2013)
33. Shifrin, M., Atar, R., Cidon, I.: Optimal scheduling in the hybrid-cloud. In: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, pp. 51–59 (2013)
34. Unuvar, M., Doganata, Y., Tantawi, A.: Configuring cloud admission policies under dynamic demand. In: Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on, pp. 313–317 (2013). DOI 10.1109/MASCOTS.2013.42
35. Vishwanath, K.V., Nagappan, N.: Characterizing cloud computing hardware reliability. In: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, pp. 193–204. ACM, New York, NY, USA (2010). DOI 10.1145/1807128.1807161

A Additional Properties of the Preservation Operator

Proof (Theorem 4 - upstream-increase) As we are focused on a two dimensional state-space we only need to prove that:

$$T_{CP}f(x_1, x_2 + 1) \leq T_{CP}f(x_1 + 1, x_2) \tag{25}$$

We examine the two possible decisions of the RHS of (25). We first examine preservation:

$$\begin{gathered}\min\{c + f(x_1, x_2 + 2), c' + f(x_1 - 1, x_2 + 2)\} \leq \\ c + f(x_1, x_2 + 2) \leq \\ c + f(x_1 + 1, x_2 + 1)\end{gathered}$$

We now examine the drop decision:

$$\begin{gathered}\min\{c + f(x_1, x_2 + 2), c' + f(x_1 - 1, x_2 + 2)\} \leq \\ c' + f(x_1 - 1, x_2 + 2) \leq \\ c' + f(x_1, x_2 + 1)\end{gathered}$$

□

Proof (Theorem 5 - submodularity) Similarly to Theorem 3, in the two dimensional case **Sub(1,2)**=**Sub**. We follow the same scheme used in Theorems 2 and 3, and examine the four cases of a_1 and a_2.

$\mathbf{a_1 = a_2 = 1}$

We apply submodularity once and get:

$$\begin{gathered}\min\{c + f(x_1, x_2 + 1), c' + f(x_1 - 1, x_2 + 1)\} + \\ \min\{c + f(x_1 + 1, x_2 + 2), c' + f(x_1, x_2 + 2)\} \leq \\ c + f(x_1, x_2 + 1) + c + f(x_1 + 1, x_2 + 2) \leq \\ c + f(x_1 + 1, x_2 + 1) + c + f(x_1, x_2 + 2)\end{gathered}$$

$\mathbf{a_1 = a_2 = 0}$

We apply submodularity once and get:

$$\begin{gathered}\min\{c+f(x_1,x_2+1),c'+f(x_1-1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c'+f(x_1-1,x_2+1)+c'+f(x_1,x_2+2)\leq\\ c'+f(x_1,x_2+1)+c+f(x_1-1,x_2+2)\end{gathered}$$

$\mathbf{a_1=0,a_2=1}$

The proof is straightforward:

$$\begin{gathered}\min\{c+f(x_1,x_2+1),c'+f(x_1-1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c+f(x_1,x_2+1)+c'+f(x_1,x_2+2)=\\ c'+f(x_1,x_2+1)+c+f(x_1,x_2+2)\end{gathered}$$

$\mathbf{a_1=1,a_2=0}$

The proof more complex and includes applying submodularity once and then sub-convexity:

$$\begin{gathered}\min\{c+f(x_1,x_2+1),c'+f(x_1-1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c+f(x_1,x_2+1)+c'+f(x_1,x_2+2)\leq\\ c+c'+f(x_1,x_2+1)+f(x_1,x_2+1)+f(x_1-1,x_2+2)-f(x_1-1,x_2+1)\end{gathered}$$

We can now apply $f\in\mathbf{SubC(1,2)}$ and get:

$$\begin{gathered}\min\{c+f(x_1,x_2+1),c'+f(x_1-1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c+f(x_1+1,x_2+1)+c'+f(x_1-1,x_2+2)\end{gathered}$$

□

Proof (Theorem 6 - sub-convexity)

$\mathbf{a_1=a_2=1}$

We apply sub-convexity once and get:

$$\begin{gathered}\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c+f(x_1+1,x_2+1)+c+f(x_1+1,x_2+2)\leq\\ c+f(x_1,x_2+1)+c+f(x_1+2,x_2+2)\end{gathered}$$

$\mathbf{a_1=a_2=0}$

We apply sub-convexity once and get:

$$\begin{gathered}\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c'+f(x_1,x_2+1)+c'+f(x_1,x_2+2)\leq\\ c'+f(x_1-1,x_2+1)+c'+f(x_1+2,x_2+2)\end{gathered}$$

$\mathbf{a_1=1,a_2=0}$

$$\begin{gathered}\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c'+f(x_1,x_2+1)+c+f(x_1+1,x_2+2)\leq\\ c+f(x_1,x_2+1)+c'+f(x_1+2,x_2+2)\end{gathered}$$

$\mathbf{a_1}=\mathbf{0},\mathbf{a_2}=\mathbf{1}$
We apply sub-convexity twice and get:

$$\begin{gathered}\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c+f(x_1+1,x_2+1)+c'+f(x_1,x_2+2)\leq\\ c+f(x_1,x_2+1)+f(x_1+2,x_2+2)-f(x_1-1,x_2+2)\\ +c'+f(x_1-1,x_2+1)+f(x_1+1,x_2+2)-f(x_1,x_2+1)=\\ c'+f(x_1-1,x_2+1)+c+f(x_1+2,x_2+2)\end{gathered}$$

□

Proof (Theorem 7 - super-convexity)
$\mathbf{a_1}=\mathbf{a_2}=\mathbf{1}$
We apply super-convexity once and get:

$$\begin{gathered}\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c+f(x_1+1,x_2+1)+c+f(x_1+1,x_2+2)\leq\\ c+f(x_1,x_2+2)+c+f(x_1+2,x_2+1)\end{gathered}$$

$\mathbf{a_1}=\mathbf{a_2}=\mathbf{0}$
We apply super-convexity once and get:

$$\begin{gathered}\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c'+f(x_1,x_2+1)+c'+f(x_1,x_2+2)\leq\\ c'+f(x_1-1,x_2+2)+c'+f(x_1+1,x_2+1)\end{gathered}$$

$\mathbf{a_1}=\mathbf{1},\mathbf{a_2}=\mathbf{0}$
This case is straightforward:

$$\begin{gathered}\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c+f(x_1+1,x_2+1)+c'+f(x_1,x_2+2)\leq\\ c+f(x_1,x_2+2)+c'+f(x_1+1,x_2+1)\end{gathered}$$

$\mathbf{a_1}=\mathbf{0},\mathbf{a_2}=\mathbf{1}$
We apply sub-convexity twice and get:

$$\begin{gathered}\min\{c+f(x_1+1,x_2+1),c'+f(x_1,x_2+1)\}+\\ \min\{c+f(x_1+1,x_2+2),c'+f(x_1,x_2+2)\}\leq\\ c'+f(x_1,x_2+1)+c+f(x_1+1,x_2+2)\leq\\ c'+f(x_1-1,x_2+2)+f(x_1+1,x_2+1)-f(x_1-1,x_2+2)\\ +c+f(x_1+1,x_2+2)\leq\\ c'+f(x_1-1,x_2+2)+c+f(x_1+2,x_2+1)\end{gathered}$$

□

B Never Preserve When $C_r > C_d$ and $C_d < C_a/(1-p_f)$

Proof (Lemma 4) Similarly to the proof of Lemma 3, for the sake of the proof completeness we first investigate the condition in a system with infinite number of servers. We note that in a system with infinite number of servers preservation can be conducted until the task completion. We examine the future operational cost the system will incur from the current preservation onwards:

$$C_a + p_f(C_a + p_f(...)) = C_a \sum_{i=0}^{\infty} (p_f)^i = \frac{C_a}{1-p_f} \tag{26}$$

Hence, in a system with infinite number of servers, for any task to be preserved we must have that the overall current and future preservations cost must be less than the drop cost. If $C_d < C_a/(1-p_f)$ then the system will never preserve a task.

We will now focus on system with finite number of servers. Obviously, in such systems there is a finite number of preservation events prior to a drop. Therefore, we write the overall system cost:

$$C_a \sum_{i=0}^{k} (p_f)^i + (p_f)^{k+1} C_d, k \geq 0 \tag{27}$$

Substituting the lemma condition in (27), we get:

$$C_a \sum_{i=0}^{k} (p_f)^i + (p_f)^{k+1} C_d > C_d, k \geq 0 \tag{28}$$

Hence, in systems with finite number of servers, if the condition holds then the system will not preserve a task. Once an admitted task server fails, the task is dropped.

□

C Always Admit when $C_r > C_d$ and $C_d \geq C_a/(1-p_f)$

Proof (Theorem 14) Similarly to Theorem 11, we will prove it by a way of contradiction. For the sake of contradiction assume that the optimal policy is policy B, that in some cases does not obey the admission rule of the theorem. Let us look at a system applying policy B (system B) and at the first such event in which system B violates the rule, that is it does not admit a new task though it has an idle server. Assume this happens at t_0.

We now construct a policy A that will perform better than policy B. System A applying policy A is identical to system B until t_0 in which it admits the new task which was rejected by system B. We denote this task as the *extra task*, and the rest of the tasks (if exist) that are in both systems as the *mutual tasks*. Similarly to sample-path technique, we couple the two systems via the failure, service and arrival times, i.e., all server failures, task departures and arrivals are the same in both systems. Once system B rejects the task the value function difference between the systems is $C_r - C_a$. We note that from t_0 system A follows system B, until at some point in time, say t_1:

1. A new task arrives and system A needs to reject it (lack of idle servers), but system B can accept the task (due to the additional idle server). Thus in this case the overall cost difference between the systems is 0. We note that from t_1 on system A imitates system B (both systems will behave similarly).

2. An active server of one of the mutual tasks fails and system A drops the task, but system B can switch to another server (due to the extra idle server). In this case system A is penalized with C_d and system B incurs server activation cost C_a. Hence, the overall cost difference is $C_r - C_d$. Also here from t_1 on both systems will behave similarly.
3. The extra task in system A is completed. In this case the difference between the systems remains $C_r - C_a$, and from this point (t_1) on both systems will behave similarly.
4. The server of the extra task in system A fails. As in System B this server is idle, this event does not impact the system and therefore system B does not incur any additional cost. In System A, we consider the case in which the task is dropped and the system is penalized with C_d. We note that due to the conditions in the theorem, i.e., $C_d \geq C_a/(1-p_f)$, this approach investigates the worst case scenario and therefore holds for all potential system trajectories. As the task is dropped, system A incurs a cost of C_d. And from this point (t_1) on both systems will behave similarly.

Similarly to Theorem 11, we again emphasize that the two systems are completely coupled before t_0, and then any time between t_0 and t_1, and then after t_1. Therefore, we cover all trajectories at which the systems differ from each other.

We note that each of the aforementioned scenarios occur in a certain probability. Due to the cost structure, it is obvious that $C_r - C_a > 0$ and $C_r - C_d > 0$. We focus on scenarios 3 and 4, which depends on the extra task and hence their probabilities are $1-p_f$ and p_f, respectively. Therefore, the overall expected cost of System A in this scenario is: $C_A = C_a + p_f C_d$; and the overall expected cost of system B is: $C_B = C_r$. We emphasize again, that dropping the extra task is the worst case scenario, i.e., incurs the highest cost. We compare the two systems cost difference ($C_A - C_B$):

$$C_a + p_f C_d - C_r \tag{29}$$

We substitute $C_d < C_r$ in (29), which yields:

$$C_A - C_B < C_a - (1-p_f)C_r \tag{30}$$

Based on the conditions of the theorem, i.e., $C_r \geq C_a/(1-p_f)$, it is clear that $C_A < C_B$. Thus, system A achieves an overall lower expected system cost than system B, considering all scenarios and their occurrence probabilities. We note that as our investigation is of an arbitrary task at an arbitrary point in time (and hence in system state) we can conclude that it is always beneficial to preserve an already admitted task (as long as there are available servers) when an active server fails.

□