Admission Control Policies for a Multi-class QoS-aware Service Oriented Architecture

Marco Abundo  Valeria Cardellini  Francesco Lo Presti
marco.abundo@gmail.com  cardellini@ing.uniroma2.it  lopresti@info.uniroma2.it

Dipartimento di Informatica, Sistemi e Produzione, Università di Roma “Tor Vergata”, 00133 Roma, Italy

ABSTRACT
In the service computing paradigm, a service broker can build new applications by composing network-accessible services offered by loosely coupled independent providers. In this paper, we address the problem of providing a service broker, which offers to prospective users a composite service with a range of different Quality of Service (QoS) classes, with a forward-looking admission control policy based on Markov Decision Processes (MDP). This mechanism allows the broker to decide whether to accept or reject a new potential user in such a way to maximize its gain while guaranteeing non-functional QoS requirements to its already admitted users. We model the broker using a continuous-time MDP and consider various techniques suitable to solve both infinite-horizon and finite-horizon MDPs. To assess the effectiveness of the MDP-based admission control for the service broker, we present simulation results where we compare the optimal decisions obtained by the analytical solution of the MDP with other admission control policies. To deal with large problem instances, we also propose a heuristic policy for the MDP solution.

Categories and Subject Descriptors
C.4 [Performance of Systems]: Modeling Techniques
; G.3 [Probability and Statistics]: Markov Processes

Keywords
Admission control, Markov Decision Process, Quality of Service, Service Oriented Architecture

1. INTRODUCTION
In the Service Oriented Architecture (SOA) paradigm, the design of complex software is facilitated by the possibility to build new applications by composing network-accessible loosely-coupled services. The so built composite service is offered by a service broker to a range of different classes of users characterized by diverse Quality of Service (QoS) requirements. The broker and its users generally engage in a negotiation process, which culminates in the definition of a Service Level Agreement (SLA) about their respective duties and QoS expectations.

In the upcoming Internet service marketplace, multiple service providers may offer similar competing services corresponding to a functional description but at differentiated levels of QoS and cost. Therefore, in undertaking the management of the SOA-based system that offers the composite service, the broker has to meet both functional requirements concerning the overall logic to be implemented and non-functional requirements concerning the QoS levels that should be guaranteed. Hence, the service broker has to select at runtime the best set of component services implementing the needed functionalities in order to maximize some utility goal (e.g., its revenue) while guaranteeing the QoS levels to the composite service users. However, the latter is a challenging task because of the highly variable nature of the SOA environment. Recently, a significant number of research efforts have been devoted to service selection issues, e.g., [2, 9, 10]. The common aim of these works is to identify for each abstract functionality in the composite service a pool (eventually a singleton) of corresponding concrete services, selecting them from a set of candidates.

However, the candidate concrete services that the service broker can use to provide the functionalities of its composite service are in a limited number. Furthermore, the service broker contracts a SLA with each concrete service provider; the set of these SLAs defines the constraints within which the broker should try to meet the QoS objectives agreed with its users and possibly to earn some revenue. Therefore, the service broker needs to apply some admission control mechanism on the users requesting to establish a SLA for the composite service. The goal is to admit only those users for which the SOA system holds sufficient resources without incurring the risk of overcommitment, and, at the same time, to exploit the available resources in a cost-effective way.

In this paper, we consider a service broker that manages a composite service offering differentiated QoS service classes to its prospective users and propose admission control policies to determine the admissibility of a user once it requests to establish a SLA for using the composite service. We formulate the admission control policy problem for the broker using Markov Decision Processes (MDPs), which provide us a rigorous formal framework. The decision to accept another user of the composite service may influence both the QoS levels perceived by that user as well as those of ongoing users already in the SOA system; moreover, this decision changes the state of the system and therefore has an impact on whether future users will be accepted.

Specifically, the MDP-based admission control policies we present in this paper are tailored to the service broker we proposed in [5, 10] and that is named MOSES, which stands for Model-based Self-adaptation of SOA systems. We model the MOSES system as a continuous-time MDP, whose solution allows to define an optimal admission control policy, where acceptance or rejection decisions are not made myopically, but they rather forecast rewards and costs associated with the future system states. The admission control policies aim at maximizing the service broker reward, while guaranteeing non-functional QoS requirements to its users. Therefore, differently from our previous work [5, 10], the broker does not carry on an altruistic strategy only aiming at satisfying the users QoS expectations, but it rather meets its targets in a selfish mode. The proposed policies allow to manage the admission control for a composite service whose workflow entails the composition pat-
terms (such as sequential, conditional, loop, and parallel) that are typical of orchestration languages such as BPEL [25], which is the de-facto standard for service workflows specification languages.

We consider admission control policies that apply infinite-horizon and finite-horizon decisions. To deal with large problem instances, we also propose a heuristic policy for the MDP solution. The basic idea of the proposed heuristic is to generate a large number of random walks of a suitable length and decide whether to admit or refuse a new user based on the sample average discounted reward under the two decisions. We analyze the performance of the MDP-based admission control policies through simulation experiments and compare them to a myopic admission control strategy. According to the latter, which is currently implemented in the MOSES prototype [5], the service broker takes admission decisions using only the present system state, on the basis of the SLA of the requesting user and the SLAs of already admitted users that are using the SOA system. To the best of our knowledge, the approach we propose in this paper is the first admission control policy based on MDPs for QoS-aware composite services.

The rest of the paper is organized as follows. In Section 2 we present the SOA system managed by the service broker. In Section 3 we describe how we have modeled the SOA system with a continuous-time MDP; we also present the heuristic we have developed to solve the MDP. Then, in Section 4 we sketch out the implementation of our admission policies and present the simulation experiments to assess the effectiveness of the proposed MDP-based approach. In Section 5 we review some related research efforts that have focused on the application of MDP-based models and stochastic programming to SOA systems and, more generally, to software systems. Finally, we draw some conclusions and give hints for future work in Section 6.

2. MOSES SYSTEM

MOSES is a QoS-driven runtime adaptation framework for SOA-based systems, designed as a service broker. The runtime adaptation features rely on the service selection strategy carried out by MOSES, which determines how to bind service requests of currently admitted users to actual component services as to optimize the broker profit. In addition, upon a user arrival, MOSES must determine whether to admit it or not to the system, since the system resources may be scarce and the broker should meet the QoS objectives agreed with its currently admitted users and possibly earn some revenue. In this section, we provide an overview on the MOSES system for which we propose in this paper MDP-based admission control policies. A detailed description of the MOSES methodology, architecture and implementing prototype can be found in [10] and [5], respectively.

![Figure 1: MOSES and its operating environment.](Image)

2.1 MOSES Model

MOSES acts as a third-party intermediary between service users and providers, performing a role of provider towards the users and being in turn a requestor to the providers of the concrete services. It advertises and offers the composite service with a range of service classes which imply different QoS levels and monetary prices. Figure 1 shows a high-level view of the MOSES environment, where we have highlighted the MOSES component on which we focus in this paper, that is the SLA Manager.

The workflow that defines the composition logic of the service managed by MOSES can include most of the BPEL structured activities: sequence, switch, while, pick, and flow [25]. Figure 2 shows an example of BPEL workflow, described as a UML2 activity diagram, that can be managed by MOSES. This example encompasses all the BPEL structured activities mentioned above, except for the pick construct.

![Figure 2: A MOSES-compliant workflow.](Image)

MOSES performs a two-fold role of service provider towards its users, and of service user with respect to the providers of the concrete services it uses to implement the composite service. Hence, it is involved in two types of SLAs, corresponding to these two roles. In general, a SLA may include a large set of parameters, referring to different kinds of functional and non-functional attributes of the service, and different ways of measuring them. MOSES presently considers the average value of the following attributes:

- **response time**: the interval of time elapsed from the service invocation to its completion;
- **reliability**: the probability that the service completes its task when invoked;
- **cost**: the price charged for the service invocation.

Other attributes, like reputation or availability, could be easily added. Our model for the SLA between the provider and the user of a service consists of a tuple \( (T, C, R, L) \), where: \( T \) is the upper bound on the average service response time, \( C \) is the service cost per invocation, \( R \) is the lower bound on the service reliability. The provider guarantees that thresholds \( T \) and \( R \) will hold on average provided that the request rate generated by the user does not exceed \( L \).

In the case of the SLAs between the composite service users and MOSES (acting the provider role), we assume that MOSES offers a set \( K \) of service classes. Hence, the SLA for each user \( u \) of a class \( k \in K \) is defined as a tuple \( (T^u_{\text{max}}, C^u_k, P^u_k, R^u_{\text{min}}, L^u_k) \). The two additional parameters \( P^u_k \) and \( R^u_{\text{min}} \) represent the penalty rates MOSES will refund its users with for possible violations of the service class response time and reliability, respectively. All these coexisting SLAs (for each \( u \) and \( k \)) define the MOSES QoS objectives. We observe that MOSES considers SLAs stating conditions that should hold for a flow of requests generated by a user.
We assume that MOSES (acting the user role) has identified for each task in the composite service a set of corresponding concrete services implementing it. We denote by $S_i$ the $i$-th task in the service composition, where $i = 1, \ldots, m$ and $m$ is the number of tasks (Figure 2 shows 6 tasks, represented by $S_1, \ldots, S_6$), and by $I_i = \{i_{ij}\}$ the set of candidate concrete services implementing $S_i$. Different concrete services can be offered by different providers with different QoS attributes. The SLA between MOSES and the provider of the concrete service $cs_{ij}$ is defined as a tuple $(l_i, c_{ij}, r_{ij}, h_i)$. These SLAs define the constraints within which MOSES should try to meet its QoS objectives.

### 2.2 MOSES Service Selection

New users requesting the composite service managed by MOSES are subject to an accept/deny decision, with which MOSES determines whether or not it is convenient to admit the user in the system according to the user SLA and the system state (present or even future, on the basis of the adopted admission control policy). We will present in Section 3 the MDP-based formulation of the admission control carried out by the SLA Manager MOSES component.

Once a user requesting a SLA has been admitted by the SLA Manager, it starts generating requests to the composite service managed by MOSES until its contract ends. Each request involves the selection of a provider from the pool of service providers that offer it. We model this selection by associating with each service class $i$ a vector $x_i = \{x_{i1}, \ldots, x_{im}\}$, where each entry $x_{ik} = \{x_{ik}^1, \ldots, x_{ik}^m\}$ is the selection policy for task $S_k$, and a given class $k$, subject to the constraints $0 \leq x_{ik}^j \leq 1$, $j \in I_i$, $\sum_{j \in I_i} x_{ik}^j = 1$. Specifically, each entry $x_{ik}^j$ of $x_i^k$ denotes the fraction of class-$k$ requests for task $S_k$ that are bound to the concrete service $cs_{ij}$. In the following, for brevity, we denote with $i$ and $j$ the task index and the concrete service index, respectively.

The service selection is driven by the solution of a suitable optimization problem. In this paper, we assume that the broker wants to maximize its profit. Let $L_k^i = \sum_{i \in I_i} L_{ik}$ denote the aggregate class-$k$ users service request rate and let $\Lambda = (\Lambda^k)_{k \in K}$. The broker profit (per unit of time) is then:

$$C(\Lambda, x) = \sum_{k \in K} \Lambda_k \left[ \sum_{i \in I_i} c_{ik} \left( \frac{x_{ik}}{x_{ik}^0} - \left( c_{ik}(\Lambda, x) + p_{ik}^\tau + p_{ik}^\rho \right) \right) \right]$$

i.e., the sum over all service classes of the service class-$k$ invocation rate $\Lambda_k$ times the per invocation reward, that is $C_{ik}$ minus the cost $C_{ik}(\Lambda, x)$ increased by the penalty $P_{ik}^\tau + P_{ik}^\rho$ for service violation; here $\tau^k$ and $\rho^k$ represent the amount of violation with respect to the agreed upon response time $T_{min}$ and reliability $R_{min}$ and $P_{ik}^\tau$ and $P_{ik}^\rho$ the penalty rates defined in the SLA.

We formulate the service selection problem as a Linear Programming (LP) maximization problem which takes the following form:

**Problem MAXWR: max $C(\Lambda, x)$**

subject to:

1. $T_k(\Lambda, x) \leq T_{max}^k + \tau^k$, $k \in K$ (2)
2. $R_k(\Lambda, x) \geq R_{min}^k - \rho^k$, $k \in K$ (3)
3. $C_k(\Lambda, x) \leq C^k$, $k \in K$ (4)
4. $l_{ij}(\Lambda, x) \leq l_{ij}$, $j \in I_i$, $1 \leq i \leq m$ (5)
5. $\sum_{j \in I_i} x_{ij} \geq 0, \sum_{j \in I_i} x_{ij} = 1$, $1 \leq i \leq m, k \in K$ (6)
6. $\tau^k \geq 0, \rho^k \geq 0$, $k \in K$ (6)

where $T_k(\Lambda, x) = \sum \frac{T_k}{\mu_k}$ is the maximum response time, $R_k(\Lambda, x) = \sum \frac{R_k}{\mu_k}$ is the reliability, $C_k(\Lambda, x)$ is the class-$k$ cost, $l_{ij}(\Lambda, x) = \sum_{j \in I_i} x_{ij}^k$ is the aggregate class-$k$ service requests made up for task $S_k$, and $\tau^k$ and $\rho^k$ represent the amount of violation with respect to the agreed upon response time $T_{min}$ and reliability $R_{min}$.

### 3. AN MDP FORMULATION FOR MOSES ADMISSION CONTROL

The service selection strategy $x$ determines how MOSES adaptively (and probabilistically) binds service requests of currently admitted users to actual services as to optimize its profit. In addition, upon a user arrival, MOSES must determine whether to admit it or not to the system. While we expect MOSES to always admit new users in a light load scenario, we expect that because of lack of resources and/or the possibility of more profitable new users in the near future, MOSES could find convenient to deny admission to some users in a heavy load scenario in order to optimize its long term profit goal.

In this section we formulate the MOSES admission control problem as a Continuous-time Markov Decision Process (CTMDP). First, in Section 3.1 we present our broker model and define the CTMDP: its state space, the broker actions/decisions, the state transition dynamics and the reward structure. In Section 3.2, we present our infinite-horizon performance criterion and how to compute the optimal policy while in Section 3.3 we present the finite-horizon version. Finally, in Section 3.4 we propose the heuristic policy.

#### 3.1 CTMDP Model

**Broker Model**

We consider a broker that has a fixed set of candidate concrete services (and associated SLAs) with which offers the composite service to prospective users. Prospective users contact the broker to establish a SLA for a given class of service $k$ and a desired contract duration. We model the arrival process for service class $k$ and contract duration of expected length $1/\mu_k$ as a Poisson process with rate $\lambda_k$. We assume that the contract durations are exponentially
distributed with finite mean $1/\mu_d > 0 \; d \in D = \{ 1, \ldots, d_{\text{max}} \}$ (which we assume for the sake of simplicity to not depend on the service class $k$). Upon a user arrival, the broker has to decide whether to admit a user or not. If a user is admitted, the user will generate a flow of requests at rate $\lambda_k$ for the duration of the contract. When a user contract expires, the user simply leaves the system. The broker set of actions is then just the pair $A = \{ a_k, a_r \}$, with $a_k$ denoting the accepting decision and $a_r$ the refusal decision.

At any given time $t$, the broker state is summarized by the broker users matrix $n_b(t) = (n_{k,d}^b(t))_{k \in K, d \in D}$, where $n_{k,d}^b(t)$ denotes the number of users for each service class $k$ at time $t$. $n_b(t)$ takes values in the set $\mathcal{N}$ of all possible broker user matrices for which the optimization problem MAXRW introduced in Section 2 has a feasible solution.

State

In Markov Decision Processes, decisions are associated to states (cfr. Ch. 2 [29]). Since in our problem decisions are associated to user arrivals we find convenient to follow an approach similar to [33] and consider a CTMDP model which incorporates the event requiring a decision into the model state. To this end, the state $s$ of our CTMDP consists of the following two components:

- the broker users matrix $n = (n_{k,d}^b)_{k \in K, d \in D}$ before the last user arrival/departure event occurred;
- the last user arrival/departure event $\omega$.

We will denote an arrival/departure event by a matrix $\omega = (\omega_k^d)_{k \in K, d \in D}$, where $\omega_d^k = 1$ if a new user makes an admission request for service class $k$ and for a contract duration with mean $1/\mu_d$, $\omega_d^k = -1$ if an existing user of class $k$ and contract duration of mean $1/\mu_d$ terminates his contract, and $\omega_d^k = 0$ otherwise. We will denote by $\Omega$ the set of all possible events.

More formally, let $t_0, t_1, \ldots$ represent the successive users arrival/departure events in the system and let $w_i$ denote the $i$-th event. Let $i(t)$ denote the number of events prior to time $t$, i.e., $i(t) = \max \{ t_i \leq t \}$. Then, at time $t$ the state is $s(t) = (n(t), \omega(t))$ with:

$$n(t) = n_b(t_{i(t)}) \text{ and } \omega(t) = \omega_{i(t)} \quad (7)$$

In other words, the CTMDP state consists of the broker state at time $t_{i(t)}$, i.e., an instant prior the last event, and the event itself. We can regard $n(t)$ as a (one event) delayed version of $n_b(t)$. Given the definition of $s(t)$ above, we can also express the broker user configuration $n_b(t)$ as function of the CTMDP state $s(t) = (n(t), \omega(t))$:

$$n_b(t) = n(t) + \omega(t)1_{i(t) \neq a_r} \quad (8)$$

where $1_{(\cdot)}$ is the indicator function.

The state space $\mathcal{S}$ consists of all possible user configuration-next event combinations, i.e.,

$$\mathcal{S} = \{ s = (n, \omega) | n \in \mathcal{N}, \omega \in \Omega, \omega_k^d \geq 0 \text{ if } n_k^d = 0 \}$$

Actions/Decisions

The broker avails itself of the possibility to accept or refuse a new user. For each state $s = (n, \omega)$, the set of available broker actions/decisions $A(s)$ depends on the event $\omega$. If $\omega$ denotes an arrival, the broker has to determine whether to accept it or not; thus $A(s) = \{ a_k, a_r \}$. If, instead, $\omega$ denotes a contract termination, there is no decision to be taken and $A(s) = \emptyset$.

<table>
<thead>
<tr>
<th>Event $\omega$</th>
<th>Decision</th>
<th>Next state $s' = (n', \omega')$</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrival</td>
<td>admitted $(a = a_k)$</td>
<td>$(n + \omega', \omega')$</td>
</tr>
<tr>
<td></td>
<td>refused $(a = a_r)$</td>
<td>$(n, \omega')$</td>
</tr>
<tr>
<td>departure</td>
<td>-</td>
<td>$(n + \omega', \omega')$</td>
</tr>
</tbody>
</table>

Transitions

System transitions are caused by users arrivals or departures and broker decisions. Given the current state $s = (n, \omega)$, the new state $s' = (n', \omega')$ is determined as follows:

- $\omega'$ is the event occurred;
- $n'$ - by definition - is the broker user configuration before the event $\omega'$, i.e., the last event, occurred. We can compute it directly from $n$ since $n'$ is just the broker user configuration after the decision $a \in A(s)$ was taken by the broker upon the previous event $\omega$. In compact form we can write $n' = n + \omega 1_{\{a \neq a_r\}}$. From (8), we observe that if the current state is $s = (n, \omega)$ the broker user configuration $n_b$ is equal to $n'$ which will characterize the next state $s'$. Table 1 summarizes all the possible transitions.

The associated transition rates are then readily obtained:

$$q_{s,s'} = \begin{cases} \lambda_k n_d^b, & \omega_d^k = 1 \\ \mu_d n_b^k, & \omega_d^k = -1 \end{cases} \quad (9)$$

Observe that $q_{s,s'}$ implicitly depend on the decision $a$ taken in state $s$ since the next state $s'$ depends on both the event $\omega'$ and $a$ (see Table 1). We find convenient to define the one-step transition probabilities for the MDP:

$$P\left(s' | s, a \right) = \frac{q_{ss'}}{\sum_{s' \in S} q_{ss'}} \quad (10)$$

where the denominator $\sum_{s' \in S} q_{ss'}$, we will also denote hereafter by $\beta(s, a)$, is the rate out of state $s$ if action $a$ is chosen, i.e.,

$$\beta(s, a) = \sum_{s' \in S} q_{ss'} = \sum_{k \in K, d \in D} (\lambda_k n_d^b + n_b^k \mu_d).$$

Rewards

To each state/decision pair $(s, a)$ is associated a reward $c(s, a)$. In our model the reward is the profit of the broker associated to the user configuration corresponding to the pair $(s, a)$ which, by (8), is $n_b = n + \omega 1_{\{a \neq a_r\}}$. Thus, we can compute $c(s, a)$ by determining the broker profit corresponding to the optimal service selection under $n_b$. To this end, we need to solve an instance of MAXRW with aggregate service request rate $\Lambda = \Lambda(s, a) = \Lambda^k(s, a)_{k \in K}$ where $\Lambda^k(s, a) = \lambda_k n_d^b$ is the aggregate class-$k$ users service request rate when the broker user configuration is $n_b$, i.e., $\Lambda^k(s, a) = n_b k \lambda_k$ and $n_d^b = \sum_{d \in D} n_d^b$ is the number of user in service class $k$. From the solution of MAXRW, we readily obtain $c(s, a)$ by taking the value of the objective function under the optimal policy $x^*(\Lambda(s, a))$, that is,

$$c(s, a) = C(\Lambda(s, a), x^*(\Lambda(s, a)). \quad (11)$$

This completes the CTMDP description. We conclude by observing that the CTMDP model dynamics is entirely driven by the user arrivals/departures and by the broker acceptance/rejection decisions. The broker logic, how currently admitted users are served, and the corresponding QoS levels are instead captured by the reward. The structure which models the broker profit. We show how to optimize it next.
3.2 Optimal Policy - Infinite Horizon

An admission control policy π for the service broker is a function π : S → A which defines for each state s ∈ S whether the broker should admit or refuse a new user. We are interested in determining the admission control policy which maximizes the broker discounted expected profit with discounting rate α > 0. For a given policy π let \( v^*_\pi(s) \) be the expected infinite-horizon discounted reward given s as initial state, defined as:

\[
v^*_\pi(s) = E_\pi^s \left\{ \sum_{t=0}^{\infty} \int_{t_i}^{t_{i+1}} e^{-\alpha u} c(s_t, a_t) \, du \right\}
\]

where \( s_t \) is the CTMDP process state at time \( t \), (up to \( t_{i+1} \)) and \( a_t \) the decision taken at time \( t_i \).

The optimal policy \( \pi^* \) satisfies the optimality equation (see 15.4 in [29]):

\[
v^*_\pi(s) = \sup_{a \in A(s)} \left\{ \frac{c(s, a)}{\alpha + \beta(s, a)} + \frac{\beta(s, a)}{\alpha + \beta(s, a)} \sum_{s' \in S} P(s'|s, a) v^*_\pi(s') \right\}, \forall s \in S
\]  \hspace{1cm} (13)

In (13), the first term \( \sup_{a \in A(s)} \frac{c(s, a)}{\alpha + \beta(s, a)} \) represents the expected total discounted reward between the first two decision epochs given the system initially occupied state \( s \) and taken decision \( a \). The second term represents the expected discounted reward after the second decision epoch under the optimal policy.

The optimal policy \( \pi^* \) can be obtained by solving the optimality equation (13) via standard techniques, e.g., value iteration, LP formulation [29]. In the case of value iteration, which we adopted in our implementation, the optimal policy is computed iteratively as follows (see 6.3.2 [29]):

**Algorithm 1: Value Iteration Algorithm**

Select an initial estimate \( v^{(0)}(s), s \in S \), specify \( \epsilon > 0 \) and set \( i = 0 \);

repeat for all \( s \in S \)

\[
v^{(i+1)}(s) \leftarrow \sup_{a \in A(s)} \left\{ \frac{c(s, a)}{\alpha + \beta(s, a)} + \frac{\beta(s, a)}{\alpha + \beta(s, a)} \sum_{s' \in S} P(s'|s, a) v^{(i)}(s') \right\}, \forall s \in S
\]

end until \( \|v^{(i+1)} - v^{(i)}\| \leq \epsilon \)

The optimal policy is then obtained by choosing for each state \( s \in S \) the decision

\[
a(s) = \text{argmax}_{a \in A(s)} \left\{ \frac{c(s, a)}{\alpha + \beta(s, a)} + \frac{\beta(s, a)}{\alpha + \beta(s, a)} \sum_{s' \in S} P(s'|s, a) v^{(i)}(s') \right\}, \forall s \in S
\]  \hspace{1cm} (15)

3.3 Optimal Policy - Finite Horizon

A potential limitation of the infinite-horizon approach we presented above arises from the curse of dimensionality which gives rise to state explosion. As shown in the next section, in our setting, even for small problem instances, we incurred high computational costs because of the large state space. As a consequence, this approach might not be feasible for online operation where a new policy must be recomputed as user statistics, e.g., the user arrival rate \( \lambda \), or the set of concrete services varies over time. In alternative, we also consider finite horizon policies, which are amenable to efficient implementations and allow to trade-off complexity vs horizon length. Furthermore, finite horizon policies also take into account the fact that in a time varying system it might not be appropriate to consider a stationary, infinite horizon policy.

In a finite-horizon setting, our aim is to optimize the expected \( N \) step finite-horizon discounted reward given \( s \) as initial state, \( v^*_m(s) \) defined as:

\[
v^*_{\pi_m}(s) = E_\pi^s \left\{ \sum_{t=0}^{N-1} \int_{t_i}^{t_{i+1}} e^{-\alpha u} c(s_t, a_t) \, du \right\}
\]

for a suitable \( N \) which defines the number of decision epochs over which the reward is computed.

For finite horizon problem, the optimal policy \( \pi^*_N \) satisfies the following optimality equation:

\[
v^*_N(s) = \sup_{a \in A(s)} \left\{ \frac{c(s, a)}{\alpha + \beta(s, a)} + \frac{\beta(s, a)}{\alpha + \beta(s, a)} \sum_{s' \in S} P(s'|s, a) v^*_N(s') \right\}, \forall s \in S
\]  \hspace{1cm} (17)

where \( v^*_N(s) \) is the expected discounted reward under policy \( \pi \) from decision epoch \( i \) to \( N \) and \( v^*_N(s) = v^*_m(s) \). The optimal policy \( \pi^*_N \) can be computed directly from (17) via backward induction by exploiting the recursive nature of the optimality equation [29].

3.4 Heuristic

Even finite horizon optimal policy evaluation might turn not to be feasible but for small size systems especially as the horizon \( N \) increases. Thus, to deal with the more general case, we need to resort to heuristics for the solution of MDP. In this section, we present the heuristic we developed and implemented for MOSES. We called it Repeated Random Walks (RRW), which is inspired by the trajectory based sparse sampling technique proposed by Pérét and García in [26]. The idea behind the RRW heuristic is quite simple. Given the current state \( s \) which requires a decision to be taken (remember that in our system a decision must be made only upon user arrival; when an admitted user leaves the system, no decision must be taken) we simply generate a large number of sample paths (or as we say random walks which is more evocative) of suitable length \( H \) and decide whether to admit or refuse a new user based on the sample average discounted reward under the two decisions.

More precisely, given the current state \( s = (n, \omega) \), where \( \omega \) is a user arrival, we generate a number \( m_n \) of random walks with acceptance as first decision as well as a number \( m_\omega \) of random walks with an initial refusal. Every random walk is made of \( H \) decision epochs. Each random walk \( p \) is given by the sequence:

\[
p = (s^{(0)}, a^{(0)}) \rightarrow (s^{(1)}, a^{(1)}) \rightarrow \ldots \rightarrow (s^{(H)}, a^{(H)})
\]

with \( s^{(i)} = (n^{(i)}, \omega^{(i)}) \) and decision \( a^{(i)} \), determined as follows.

- \( s^{(0)} = s \) while \( a^{(0)} = a_a \) or \( a^{(0)} = a_r \), depending whether we are simulating a random walk with initial acceptance or refusal. For \( i = 1, \ldots, H \):
s(i) is randomly determined from state s(i−1) according to the one step transition probabilities:

\[ P(s(i) = s'| s(i−1) = s, a(i−1) = a) = \frac{q_{ss'}}{\sum_{s' \in S} q_{ss'}}. \]  

(18)

\[ a(i) \] is greedily chosen as the decision which maximizes the immediate reward induced by the action, i.e., by considering the reward during step-i alone, which simply reduces to:

\[ a(i) = \arg\max_{a} \frac{c(s(i), a)}{\alpha + \beta(s(i), a)} \]  

(19)

The expected reward \( r_p \) for a random walk \( p \) (observe that while the sequence of state/decisions is fixed in a given random walk, the time spent in each state during each step is still a random variable) can be obtained by 13.3.6 [29] with \( r_p = r_{p,H} \), where \( r_{p,H} \) obeys the following recursion:

\[ r_{p,H} = \frac{c(s(H−h), a(H−h))}{\alpha + \beta(s(H−h), a(H−h))} + \sum_{h=0}^{H−1} \beta(r_{a(h)}, c(s(h), a(h))) \]  

(20)

where

\[ \beta_{a(h)} = \prod_{j=0}^{h} \frac{\beta(s(j), a(j))}{\alpha + \beta(s(j), a(j))} = \frac{\beta(s(0), a(0))}{\alpha + \beta(s(0), a(0))} \]  

(22)

Let \( v_{i,a}, i \in \{1, 2, ..., m_{a}\} \) be the total discounted reward during the i-th random walk with an initial acceptance and let \( v_{i,a} \), \( i \in \{1, 2, ..., m_{a}\} \) be the total discounted reward during the i-th random walk with an initial refusal. The average random values associated to both decisions are:

\[ \bar{v}_{a} = \frac{\sum_{i=1}^{m_{a}} v_{i,a}}{m_{a}} \]  

Finally, the optimal decision is determined by:

\[ a^{*} = \arg \max_{a \in \{1, a_{i}\}} \frac{\sum_{i=1}^{m_{a}} v_{i,a}}{m_{a}} \]  

(24)

As far as \( m_{a} \) and \( m_{a} \) are concerned, they are not fixed, but they are dynamically determined. Minimal values \( m_{a} \) and \( m_{a} \) are specified, as well as maximal values \( m_{a} \) and \( m_{a} \).

Let \( s_{n−1,a} \) be the estimation of the variance of random walks with initial decision \( a \), defined as

\[ s_{n−1,a} = \sum_{i=1}^{m_{a}} \frac{(v_{i,a} - \bar{v}_{a})^2}{m_{a} - 1} \]  

and let \( RSD_{a}^{2} \) be the estimated quadratic coefficient of variation (or quadratic relative standard deviation) given by

\[ RSD_{a}^{2} = \frac{s_{n−1,a}}{v_{a}} \]  

(26)

Then, further random walks with initial decision \( a \) are repeated until either:

\[ m_{a} \geq m_{a} \] and the quadratic coefficient of variation settles on constant value, or

\[ m_{a} \] random walks were performed.

Nevertheless, to allow on-line admission control decisions, we need to trade-off accuracy, i.e., larger \( m_{a} \) and convergence of \( RSD_{a}^{2} \) with computational cost since decisions should be taken in a timely manner. As shown in the next section, the heuristic we propose allows to achieve close to optimal results within few seconds, which we believe is compatible with on-line operations of a SOA system.

4. EXPERIMENTAL ANALYSIS

In this section, we present the experimental analysis we have conducted through simulation to assess the effectiveness of the MDP-based admission control for MOSES. We first describe the simulation model and then present the simulation results.

4.1 Simulation Model

Following the broker model in Section 3, we consider an open system model, where new users belonging to a given service class \( k \in K \) and expected contract duration \( 1/\mu_k \) arrive according to a Poisson process of rate \( \lambda_k \). We also assume exponential distributed contract duration. Once a user is admitted, it starts generating requests to the composite service according to an exponential inter-arrival time with rate \( \lambda_k \). Since SOA workload characterization has been not deeply investigated up to now, we assume to have Poisson arrivals (that can be generalized to Markov arrivals) and exponential contract durations. These assumptions allows us to take advantage of Markov models and to provide our admission control with a solid theoretical basis.

The discrete-event simulator has been implemented in C language using the CSIM 20 tool [22]. Multiple independent random number streams have been used for each stochastic model component. The experiments involved a minimum of 10,000 completed requests to the composite service; for each measured mean value the 95% confidence interval has been obtained using the run length control provided by CSIM. As regards the admission control policies, they have been implemented in MATLAB.

4.2 Experimental Results

We illustrate the dynamic behavior of our admission control policies assuming that MOSES provides the composite service whose workflow is shown in Figure 2. For the sake of simplicity, we assume that two candidate concrete services (with their respective SLAs) have been identified for each task, except for \( S_2 \) for which four concrete services have been identified. The respective SLAs differ in terms of cost \( c \), reliability \( r \), and response time \( t \) (being the latter measured in sec); the corresponding values are reported in Table 2 (where \( i,j \) denotes the concrete service). For all concrete services, the maximum load \( l_{ij} \) is set to 10 invocations per second.

<table>
<thead>
<tr>
<th>( t )</th>
<th>( l_{ij} )</th>
<th>( l_{ij} )</th>
<th>( l_{ij} )</th>
<th>( l_{ij} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>1.1</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>1.2</td>
<td>1.2</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>2.1</td>
<td>2.1</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>2.2</td>
<td>2.2</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>2.3</td>
<td>2.3</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
</tr>
<tr>
<td>3.1</td>
<td>3.1</td>
<td>0.995</td>
<td>0.995</td>
<td>0.995</td>
</tr>
</tbody>
</table>

Table 2: Concrete service SLA parameters.
On the user side, we assume a scenario with four service classes (i.e., $1 \leq k \leq 4$) of the composite service managed by MOSES. The SLAs negotiated by the users are characterized by a wide range of QoS requirements, with users in class 1 having the most stringent performance requirements and highest cost paid to the broker, and users in class 4 the least stringent performance requirements and lowest cost. In the first scenario, $C^k = 18$, $R^k_{\text{min}} = 0.9$, $T^k_{\text{max}} = 11$; class 2: $C^k = 25$, $R^k_{\text{min}} = 0.95$, $T^k_{\text{max}} = 7$; class 3: $C^k = 15$, $R^k_{\text{min}} = 0.9$, $T^k_{\text{max}} = 15$; class 4: $C^k = 12$, $R^k_{\text{min}} = 0.85$, $T^k_{\text{max}} = 18$. The penalty rates $P^k_c$ and $P^k_\beta$ are set equal to the reciprocal of the corresponding SLA parameter.

Furthermore, for each service class we consider two possible contract durations (i.e., $d_{\text{max}} = 2$), which can be either short or long. Therefore, the system state $s = (n, \omega)$ is characterized by a $4 \times 2$ brokers users matrix $n$, as defined in Section 3.1.

In the following, we analyze two different sets of experiments. In the first set we evaluate the effectiveness of the MDP infinite horizon policy, while in the second set we evaluate the proposed heuristic and study its performance vs computational complexity tradeoff. For space limits, we omit the results of the finite horizon policy. In our experiments, we found out that the finite horizon policy yields results almost identical to the heuristic (under a finite horizon $N$ equal to the trajectory length $H$ of the heuristic) but at a higher computational cost.

### 4.2.1 Infinite Horizon Policy

In the first set of experiments, we compare the proposed MDP-based infinite horizon policy with the simple policy currently implemented in the MOSES prototype [5, 10], which we refer to as the blind policy. Under the infinite horizon policy, the admission control decisions are based on the optimal policy $\pi^*$, which is obtained by solving the optimality equation (13) via the value iteration method, setting the discount rate $\alpha = -\ln(0.9) \approx 0.1$ and the parameter $\epsilon = 0.01$. For the blind policy, no reasoning about future rewards is considered, and a user is admitted if the composite service as long as enough resources are available to serve its requests at the required QoS level without violating existing users QoS. More precisely, in the blind policy a new is user is accepted if the service selection optimization problem MAXRW in Section 2.2 can be solved given the SLA requested by the new users and the SLAs agreed by MOSES with its currently admitted users. Specifically, the user request rate $L^k_n$ is added to the aggregate flow $\lambda^k$ of class-$k$ requests currently served by MOSES, and the so obtained instance of the LP optimization problem is solved. If a solution exists, the user is admitted; otherwise, its SLA request is rejected.

We consider two different scenarios, where we vary the arrival rate of the contract requests. On the other hand, in both scenarios the amount of requests generated by an admitted user is $L^k_n = 1$ req/sec and the contract duration is fixed to $(1/\mu_d)_{d \in D} = (50, 200)$, where the first component corresponds to shorter contracts and the latter to longer contracts. In the following, we will denote short and long contracts with $s$ and $l$, respectively.

In the first scenario, we set the matrix $(\lambda^k_{n})_{k \in K, d \in D} = \begin{pmatrix} 0.02 & 0.02 & 0.02 & 0.02 \\ 0.02 & 0.02 & 0.02 & 0.02 \\ 0.02 & 0.02 & 0.02 & 0.02 \\ 0.02 & 0.02 & 0.02 & 0.02 \end{pmatrix}$, that is, all the contract requests arrive at the same rate, irrespectively of the service class. In the second scenario, $(\lambda^k_{n})_{k \in K, d \in D} = \begin{pmatrix} 0.02 & 0.02 & 0.04 & 0.04 \\ 0.02 & 0.02 & 0.04 & 0.04 \\ 0.02 & 0.02 & 0.04 & 0.04 \\ 0.02 & 0.02 & 0.04 & 0.04 \end{pmatrix}$, that is, contract requests for service classes 3 and 4, which are less profitable for the broker as their SLAs have lower $C^k$, arrive at a double (class 3) or quadruple (class 4) rate with respect to requests for service classes 1 and 2.

To compare the performance of the two admission control policies, we consider as main metrics the average reward per second of the service broker over the simulation period and the percentage of rejected contract requests. Furthermore, we also analyze the mean execution time.

Table 3 shows the average reward per second earned by the service broker for the various admission control policies and under the different considered scenarios. As expected, the infinite horizon policy maximizes the broker reward, achieving a largely significant improvement over the blind policy under all scenarios.

Let us now analyze the performance metrics separately for each scenario. From Table 3 we can see that in the first scenario the improvement achieved by infinite horizon policy over blind is equal to almost 65%. Figure 3(a) shows the percentage of rejected SLA contracts for all the service classes, distinguishing further between short and long contract durations, achieved by the different admission control policies. For each policy, the first four bars (1s, 2s, 3s, and 4s) regard the short-term contracts for the various service classes, while the latter four (1l, 2l, 3l, and 4l) the long-term ones.

![Figure 3: Rejected contract requests under both scenarios.](image-url)

While the blind policy is not able to differentiate among the service classes, the MDP-based policy favors users of the more profitable classes 1 and 2, which pay more for the composite service, over the less profitable ones which are more likely to be rejected. We also observe that the infinite horizon policy slightly differentiates between short-term and long-term contracts, favoring the latter ones. This is expected, since long-term duration contracts yield higher average reward than shorter ones.

For the second scenario, which is characterized by a higher contract request arrival rate for classes 3 and 4, as reported in Table 3,
the MDP-based policy allows the broker to more than double its revenue with a 162% increase with respect to the blind policy. Figure 3(b) shows that, as expected, both the admission control policies reject a higher percentage of contract requests for these classes with respect to the first scenario. As before, the MDP-based admission control favors service classes 1 and 2 with respect to 3 and 4, since the former ones let the system achieve higher rewards while the latter, that could use the limited system resources with a low revenue for the broker, incur in a very high refusal percentage (almost total for class 4, which is the least advantageous one). Again, long term contracts are preferred to shorter ones.

We found that once a contract request has been accepted, the QoS levels specified in the SLAs are quite largely met by MOSES for each flow of service class, independently on the applied admission policy. Figures 4(a) and 4(b) show that, notwithstanding the profit achieved through the forward-looking admission control is far greater, the percentages of QoS violations are almost the same observed when the blind policy is applied and, in some cases, even lower. The amount of violations experienced by the accepted users of a certain service class is influenced both by the SLA parameters and the fulfillment of the QoS levels for the other service classes. Figures 4(a) and 4(b) show that, for classes 1 and 2, the QoS requirements on response time are violated more often and those on cost less often, since these classes are more stringent for times and less stringent for costs. On the other hand, as the reward rates are quite similar to each other and, surprisingly, SLAs are better met for the classes which guarantee a higher reliability. We also observe that violations on costs are likely to be relatively more frequent owing to the nature of the geometric distribution related to the number of times the loop in the BPEL workflow of Figure 2 is executed.

We have chosen these parameters after many tests and their analysis: $RSD_{a,max}^2 = 0.1$, where $RSD_{a,max}^2$ is the maximum acceptable threshold for the quadratic relative standard deviation of values associated to random walks starting with action $a$, $a \in \{a_{t}, a_{d}\}$; $m_{min} = 501$ and $m_{max} = 2500$.

We have chosen these parameters after many tests and their analysis: $RSD_{a,max}^2$ is small enough to achieve a good accuracy, $m_{min}$ avoids not significant results, and $m_{max}$ prevents the RRW heuristic execution time from being too long.

The results are summarized in Tables 4 and 5, where we report the average reward per second and the execution time for different values of the sample path trajectory length $H$. For the sake of readability, we omitted the confidence intervals. In these experiments, we also varied the discount factor $\alpha$ to study its impact on both the performance and the policy computation execution time.

Overall, the heuristic performs very well. As expected, its performance improves as the trajectory length $H$ increases and reward close to the optimal value are attained even for very small values of $H$: in these experiments, even for $H = 1$ there are significant improvements with respect to the blind policy, notwithstanding that only one step in the future is considered. Furthermore, its computation requires only few seconds in the worst case and thus makes this solution amenable for online implementation.

We observe that as $\alpha$ decreases, the heuristic performance degrades both in term of average reward and execution time. This can be explained by observing that smaller discount rates imply that the interval over which reward is optimized decreases. Indeed, observe from (13) that since the reward contribution becomes negligible when $e^{-\alpha t} \ll 1$, the smaller $\alpha$ the larger the interval. On the other hand, the heuristic can only account for a limited time interval, which increases with $H$, but is nevertheless limited. Thus, the approximation introduced by the heuristic increases as $\alpha$ decreases. In these examples, the average reward that can be attained by the heuristics for $H = 2$ when $\alpha = 0.1$ requires a tenfold increase in heuristics trajectory length $H$ for $\alpha = 0.003$.

For completeness, in Table 5 we also reported the MDP optimal policy computation time for the different $\alpha$. We can observe that even for the optimal policy the computation time rapidly increases as $\alpha$ decreases. We verified that this is due to the much slower problem MAXRWR, that took 0.0638 sec on average. For the MDP infinite horizon policy, we precomputed the solution of the above optimization problem once for all. The state space generation alone (877716 states in this simple scenario) requires 233 sec; 5502 sec are needed for the matrix generation (observe that the entries of the matrix include the term $e(s, a)$, $a \in S$, $a \in A(s)$, which by itself requires the solution of the service selection optimization problem, see (11)), and 800 sec for the value iteration to converge. The computational cost thus appears to be acceptable only under stationary conditions as the policy needs to be computed only once. Unfortunately, we expect that in a real world scenario the system cannot be assumed to be stationary, since user statistics and/or the set of concrete services might unpredictably change over time. In this case, we would need to recompute the policy as changes in the system parameters are detected. This could result into excessive computational cost which might prevent the adoption of the proposed MDP-based policy for online system management.

### 4.2.2 RRW Heuristic

We ran several experiments to assess the performance of the heuristic policy with respect to the optimal policy. In all the experiments we used the following parameters:

- $RSD_{a,max}^2 = 0.1$, where $RSD_{a,max}^2$ is the maximum acceptable threshold for the quadratic relative standard deviation of values associated to random walks starting with action $a$, $a \in \{a_{t}, a_{d}\}$;
- $m_{min} = 501$ and $m_{max} = 2500$.

We have chosen these parameters after many tests and their analysis: $RSD_{a,max}^2$ is small enough to achieve a good accuracy, $m_{min}$ avoids not significant results, and $m_{max}$ prevents the RRW heuristic execution time from being too long.

The results are summarized in Tables 4 and 5, where we report the average reward per second and the execution time for different values of the sample path trajectory length $H$. For the sake of readability, we omitted the confidence intervals. In these experiments, we also varied the discount factor $\alpha$ to study its impact on both the performance and the policy computation execution time.

Overall, the heuristic performs very well. As expected, its performance improves as the trajectory length $H$ increases and reward close to the optimal value are attained even for very small values of $H$: in these experiments, even for $H = 1$ there are significant improvements with respect to the blind policy, notwithstanding that only one step in the future is considered. Furthermore, its computation requires only few seconds in the worst case and thus makes this solution amenable for online implementation.

We observe that as $\alpha$ decreases, the heuristic performance degrades both in term of average reward and execution time. This can be explained by observing that smaller discount rates imply that the interval over which reward is optimized decreases. Indeed, observe from (13) that since the reward contribution becomes negligible when $e^{-\alpha t} \ll 1$, the smaller $\alpha$ the larger the interval. On the other hand, the heuristic can only account for a limited time interval, which increases with $H$, but is nevertheless limited. Thus, the approximation introduced by the heuristic increases as $\alpha$ decreases. In these examples, the average reward that can be attained by the heuristics for $H = 2$ when $\alpha = 0.1$ requires a tenfold increase in heuristics trajectory length $H$ for $\alpha = 0.003$.

For completeness, in Table 5 we also reported the MDP optimal policy computation time for the different $\alpha$. We can observe that even for the optimal policy the computation time rapidly increases as $\alpha$ decreases. We verified that this is due to the much slower...
convergence of the value iteration algorithm for smaller $\alpha$. This confirms that for online operativeness, the proposed heuristic is to be preferred to the MDP optimal policy.

5. RELATED WORK

A considerable number of research efforts have focused on the application of MDP-based models and stochastic programming to SOA systems and, more generally, to software systems.

Some of these works have proposed self-healing approaches to support the reconfiguration of running software systems, where the user or an intermediate entity, according to available observations about the system performance, may replace or tune a system component before it fails. In [4], Beckmann and Subramanian analyzed the reliability of a system which consists of components connected in series and formulated an optimal replacement policy by means of dynamic programming. In [19], Hu and Yue presented an optimal replacement problem of a system in a semi-Markov environment. In [12], Chen and Feldman studied a repair/replacement problem formulated as a MDP; the optimal policy is obtained via analytical minimization of the total expected discounted cost. In [27], Pfening et al. analyzed how to determine the optimal time to rejuvenate a server-side software affected by age-based soft failures. In [15,18], semi-Markov processes are exploited to find the optimal software rejuvenation schedule which minimizes the total discounted cost over an infinite time horizon. In [20], the focus is on rejuvenating the application periodically and restarting it at a previous checkpoint to increase the probability of successful termination.

As regards the reconfiguration of SOA systems, in [28] Pillai and Narendra addressed self-healing issues by proposing an MDP formulation to decide about replacement of an atomic or component service that is expected to become faulty. However, their proposal considers only sequential patterns in the composite service, while we consider a significant subset of structured orchestration patterns, including sequential, conditional, parallel, and loop patterns.

Focusing on the application of MDP to the SOA field, some recent approaches [11,16,32] have used MDP to model service composition. Their aim is to create automatically an abstract workflow of the service composition that satisfies functional and non-functional requirements, and also to allow the composite service to adapt dynamically to a varying environment [32].

Some works have proposed MDP-based admission control in service-oriented systems [3,7] and are therefore most closely related to ours. In [3] Bannazadeh and Leon-Garcia proposed an admission control for service-oriented systems which uses a two-step approach for maximizing the system revenue. In [7] Bichler and Setzer applied an MDP-based formulation to tackle admission control for on demand media services. However, neither of these works considers composite services organized according to some business logic, while our approach is able to deal with composite service workflows that entail the composition patterns typical of orchestration languages.

Finally, MDPs have been extensively used to design call admission control at session level in wired and wireless networks, (e.g., [13,24,33] to name a few) and more generally to control communication systems (see [1] for a survey).

Turning our attention to MDP solution techniques, there is a considerable literature which addresses the solution of large MDPs by means of heuristics/approximations, e.g., [6,8,21,23,26,31].

Several approaches have been developed by Artificial Intelligence and Machine Learning research communities to efficiently compute sub-optimal actions, such as reinforcement learning [6,31] and structure based methods [8]. However, when the state space is huge, simulations over sample states and rewards have to be considered, as well as online stochastic search. These techniques can also be applied in the case of initially unknown MDPs by either optimizing the state value function in a direct manner [31], or by first estimating the transition and reward models and then identifying a policy resulting from a planning problem, e.g. [14,17,30].

Most of the known techniques resort to some sort of sampling. The sparse sampling approach [21] replaces myopic estimates of the value of exploration with explicit lookahead to the effective horizon. Notwithstanding the massive computational cost required, the approach is interesting and is also applicable to infinite state spaces. In [23], Mundhenk et al. proposed to control branching factor and lookahead depth.

In [26], P´eret and Garcia proposed an appealing trajectory-based strategy, which follows trajectories of length $H$ from the current state $s_t$ to a leaf state $s_{t+H}$. The global width $C$ is no more set and depends on state $s$ and action $a$. It is necessary to obtain a good trade-off between horizon $H$ and branching factor $C$, which resembles the trade-off between exploration and exploitation. Surprisingly, increasing the horizon with a fixed width might even lead to an error increase, since the term due to finite sampling approximation grows with $H$. Such a phenomenon, known as lookahead pathology, has been widely studied in game theory: deeper searches do not automatically improve the quality of decisions.

6. CONCLUSIONS

In this paper, we studied the admission control problem for a service broker, MOSES, which offers to prospective users a composite service with different QoS levels. We formulated the admission control problem as a Markov Decision Process with the goal to maximize the broker discounted reward, while guaranteeing non-functional QoS requirements to its users. We considered both

<table>
<thead>
<tr>
<th>$H = 1$</th>
<th>$H = 2$</th>
<th>$H = 5$</th>
<th>$H = 10$</th>
<th>$H = 15$</th>
<th>$H = 20$</th>
<th>Inf-horizon</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0.1$</td>
<td>39.61</td>
<td>63.00</td>
<td>86.41</td>
<td>66.70</td>
<td>66.50</td>
<td>66.70</td>
</tr>
<tr>
<td>$\alpha = 0.03$</td>
<td>39.22</td>
<td>62.38</td>
<td>66.27</td>
<td>66.90</td>
<td>66.99</td>
<td>67.08</td>
</tr>
<tr>
<td>$\alpha = 0.003$</td>
<td>55.78</td>
<td>61.72</td>
<td>63.35</td>
<td>61.20</td>
<td>63.24</td>
<td>64.85</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$H = 1$</th>
<th>$H = 2$</th>
<th>$H = 5$</th>
<th>$H = 10$</th>
<th>$H = 15$</th>
<th>$H = 20$</th>
<th>Inf-horizon</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0.1$</td>
<td>0.0021</td>
<td>0.40</td>
<td>1.14</td>
<td>2.48</td>
<td>3.93</td>
<td>5.12</td>
</tr>
<tr>
<td>$\alpha = 0.03$</td>
<td>0.0021</td>
<td>0.40</td>
<td>1.14</td>
<td>3</td>
<td>7.43</td>
<td>15.96</td>
</tr>
<tr>
<td>$\alpha = 0.003$</td>
<td>0.0021</td>
<td>0.40</td>
<td>1.19</td>
<td>5.68</td>
<td>16.26</td>
<td>23.59</td>
</tr>
</tbody>
</table>
infinite-horizon and finite-horizon cost functions and also proposed a simple and computationally efficient heuristic. We compared the different solutions through simulation experiments. Our results show that the MDP-based policies can guarantee much higher profit to the broker while satisfying the users QoS levels with respect to a simple myopic policy, which accepts users as long as the broker has sufficient resources to serve them. We also found that the heuristic achieves near to optimal performance at a fraction of the computational cost, which makes it amenable to online implementation.

Future work includes the implementation of the MDP-based admission control in the MOSES prototype and its experimental evaluation in realistic scenarios. We also intend to remove the assumption of Poisson arrivals and exponential behavior and to run simulations with different random distributions, possibly taken from some new workload characterization study of SOA traffic.

7. ACKNOWLEDGEMENTS

We would like to thank the anonymous referees for their helpful comments and Vincenzo Grassi for valuable discussions about the MDP formulation.

8. REFERENCES


